IBM WebSphere Enterprise Service Bus

Using IBM WebSphere Enterprise Service Bus

Version 7 Release 5.1

SC34-7236-01

Note

Before using this information and the product it supports, read the information in "Notices" on page 1139.

This edition applies to version 7, release 5, modification 1 of WebSphere Enterprise Service Bus (product number 5724-I82) and to all subsequent releases and modifications until otherwise indicated in new editions.

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

© Copyright IBM Corporation 2007, 2011.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

PDF books and the information center

PDF books are provided as a convenience for printing and offline reading. For the latest information, see the online information center.

As a set, the PDF books contain the same content as the information center.

The PDF documentation is available within a quarter after a major release of the information center, such as Version 6.0 or Version 6.1.

The PDF documentation is updated less frequently than the information center, but more frequently than the Redbooks®. In general, PDF books are updated when enough changes are accumulated for the book.

Links to topics outside a PDF book go to the information center on the Web. Links to targets outside a PDF book are marked by icons that indicate whether the target is a PDF book or a Web page.

Table 1. Icons that prefix links to topics outside this book

Icon	Description

	A link to a Web page, including a page in the information center.
	
	Links to the information center go through an indirection routing service, so that they continue to work even if target topic is moved to a new location.
	
	If you want to find a linked page in a local information center, you can search for the link title. Alternatively, you can search for the topic id. If the search results in several topics for different product variants, you can use the search result **Group by** controls to identify the topic instance that you want to view. For example:
	1. Copy the link URL; for example, right-click the link then select `Copy link location`. For example: `http://www14.software.ibm.com/webapp/wsbroker/redirect?version=wbpm620&product=wesb-dist &topic=tins_apply_service`
	2. Copy the topic id after `&topic=`. For example: `tins_apply_service`
	3. In the search field of your local information center, paste the topic id. If you have the documentation feature installed locally, the search result will list the topic. For example:
	`1 result(s) found for`
	
	`Group by: None | Platform | Version | Product`
	`Show Summary`
	
	`Installing fix packs and refresh packs with the Update Installer`
	
	4. Click the link in the search result to display the topic.

	A link to a PDF book.

© Copyright IBM Corp. 2007, 2011

Contents

PDF books and the information center iii

Figures xi

Tables xvii

Chapter 1. Product overview for WebSphere ESB 1

WebSphere ESB and the enterprise service bus . . . 1
What's new in this release 2
Architectural overview of WebSphere ESB 2
Service-oriented architecture core 3
Service applications and service modules 7
Imports and import bindings 8
Exports and export bindings. 10
Mediation modules 11
Mediation primitives 13
Dynamic routing 18
Mediation policy control of service requests . . 18
WebSphere Service Registry and Repository . . 19
Deployment environments in WebSphere ESB . . . 20
Business Space powered by WebSphere 21
WebSphere Adapters 23
Development and deployment of applications on WebSphere ESB 24
Migration to WebSphere ESB 25
Administration of applications on WebSphere ESB 25
Administrative control of mediation processing 26
Security on WebSphere ESB 27
System monitoring on WebSphere ESB 27
Standards compliance 28
Accessibility 28
Federal Information Processing Standards . . . 30
Internet Protocol Version 6 30
Globalization 30
Transforming string objects from one bidirectional language format to another. . . . 32
Transforming data objects from one bidirectional language format to another 33

Chapter 2. Technical overviews of WebSphere ESB. 37

Service Component Architecture 37
SCA and service invocation 37
Modules 38
Service components 40
Stand-alone references 44
Service data objects. 44
Business objects 45
Defining business objects 45
Working with business objects 46
Special business objects 48
Business object parsing mode 49
Bindings 53
Export and import binding overview 55
Export and import binding configuration . . . 58
Interoperability between SCA modules and Open SCA services 69
Binding types. 72
Relationships 157
Relationship service 159
Relationship manager 160
Relationships in Network Deployment environments 161
Relationship service APIs 161
Deployment architecture. 161
Cells 161
Servers 162
Profiles 163
Deployment managers 164
Nodes 164
Node agents. 165
Enterprise service bus messaging infrastructure . . 165
Messaging or queue destination hosts 166
Data sources for WebSphere ESB 166
Service integration buses for WebSphere ESB 168
Message Service clients 169

Chapter 3. Scenarios for Process Server and WebSphere Enterprise Service Bus 171

Introduction: ESB scenarios and use cases 171
SOA considerations 172
Flexibility and reuse 173
Connectivity. 177
Governance 181
Security 182
Robustness 187
Management 189
Connectivity scenarios 192
Overview of imports and exports. 193
Overview of adapters 195
How to connect to WebSphere MQ 196
How to connect to databases 198
How to connect to SAP XI 200
How to connect to CICS. 202
How to connect to IMS 205
Service gateway scenarios 206
Patterns for service gateways and proxies . . . 208
Overview of proxy gateways 209
Mediation policy scenarios 212
Overview of mediation policies 214
Example: Mediation policies associated with modules 216
Example: Mediation policies associated with target services 218
Service level agreement scenarios. 219
Invocation check based on an SLA between a client and a service provider 221

© Copyright IBM Corp. 2007, 2011

Endpoint selection with default configuration 223
Endpoint selection with enhanced selection criteria 225
An ESB in a stock quote scenario 227
Stock quote scenario: flexibility and reuse . . . 229
Stock quote scenario: governance 232
Using an ESB to route and transform messages . . 233
Building a service gateway 233
Dynamic routing 245
Mediation policy control of service requests . . 255
Protocol conversion 259
Validating messages 261
Connect to CICS or IMS services using MQ . . 263

Chapter 4. Tutorials for WebSphere ESB 269

Advanced tutorial: Creating mediation policies for modules 269
Introduction 270
Loading an SCA module and associated mediation policies 271
Viewing a mediation policy 272
Creating your own mediation policy, for an SCA module 275
Attaching a mediation policy to an SCA module 276
Creating a conditional mediation policy . . . 277
Governing a mediation policy 279
Tutorial summary 282
Tutorial: Building clustered topologies in WebSphere ESB. 283
Creating advanced deployment manager profile using the Profile Management Tool 283
Start the deployment manager. 291
Creating custom profiles (managed nodes) for the deployment manager using PMT 292
Federate the nodes into the deployment manager shell 296
Creating deployment environments using the deployment environment wizard 297
Tutorials about administering the server in Business Space 305
Tutorial: Creating mediation policies for modules, using Business Space 305
Tutorial: Creating mediation policies for services, using Business Space 316
Tutorial: Administering proxy gateways using Business Space 330
Tutorial: Assessing the health of your solution 338
Tutorial: Managing your timetables 339
Tutorial: Configure WSRR for use with SLA Endpoint Lookup mediation primitive 340
Introduction 342
Importing the WSDLs associated with the target services 343
Creating a service version corresponding to the WSDL Description Imported 343
Creating a service level definition (SLD) . . . 343
Creating an SLA 345
Associating the SLD and SLA with the service version 347
Setting the service endpoint to be online and available 348
Advanced: Extending the SLA Endpoint Lookup query 350
Tutorial summary 353

Chapter 5. Administering the IT infrastructure 355

How do I...? Administer the IT infrastructure for WebSphere ESB. 355
Overview of administering WebSphere ESB . . . 356
The administrative console 356
Command-line tools, scripts, and programming interfaces 362
Configuration information 363
Configuring the business object parsing mode . . 364
Viewing the Business Object Framework Parsing Mode using the administrative console 364
Viewing the Business Object Framework Parsing Mode using a command. 364
Getting started with the administrative console . . 365
Starting and stopping the administrative console 365
Setting administrative console preferences . . . 367
Setting administrative console filters. 368
Using My tasks. 368
Accessing product information and help from the administrative console 369
Accessing command assistance from the administrative console 370
Managing the administrative architecture 372
Starting deployment managers 372
Stopping a deployment manager 372
Starting node agents 373
Stopping a node agent 374
Restarting a node agent 374
Starting and stopping deployment environments 374
Starting a cluster 378
Stopping a cluster 379
Administering servers 380
Starting stand-alone servers 380
Stopping stand-alone servers 381
Starting managed servers 382
Stopping managed servers 383
Administering deployment environments 384
Modifying the deployment topology. 385
Deleting deployment environment definitions using the command line 388
Renaming a deployment environment definition using the command line 389
Removing nodes from a deployment environment definition using the command line . 390
Renaming nodes in a deployment environment definition using the command line 392
Modifying deployment environment definition parameters 393
Managing deployment environment resources 394
Editing the data source configuration 396
Stopping and restarting the deployment manager 397
Stopping and restarting a cluster member . . . 398
Starting and stopping deployment environments 399

Exporting deployment environment definitions using the administrative console 400
Exporting deployment environment definitions using the command line 401
Importing deployment environment definitions based on design documents using the administrative console 402
Importing deployment environment definitions using the command line 408
Removing deployment environments 409
Administering applications and application services 410
Administering service applications and service modules 410
Administering access to WSRR 492
Administering relationships 497
Administering the relationship service 518
Administering loading of mediation flows . . . 521
Administering Common Event Infrastructure . . . 522
Administering the event service using the administrative console 522
Administering the event service using scripting 525
Logging and tracing Common Event Infrastructure components 532
Maintaining the event database 533
Changing the configuration topology 539
Changing the configuration topology using the command line 539
Working with adapters 545
Differences between WebSphere Adapters and WebSphere Business Integration Adapters . . . 545
WebSphere adapters 548
WebSphere Business Integration Adapters . . . 548
Working with events 549
Processing events in sequence 550
Managing failed events in WebSphere ESB . . 563
Preventing multiple failures when a service is unavailable 582

Chapter 6. Developing for WebSphere ESB 593

Enterprise Service Bus programming 593
Introduction 593
Common Usage Patterns 595
Service Component Architecture 601
Mediation Flow Component 607
Routing Messages Within a Mediation Flow . . 613
Data Transformation 618
Invocation of Services 631
Service gateway 650
Aggregation 665
Error handling in the mediation flow component 683
Testing and Debugging 691
Service Component Architecture programming . . 697
Service Component Definition Language . . . 697
SCA programming model fundamentals . . . 706
SCA programming techniques 736
Business objects programming 740
Programming model 740
Programming using business object services . . 761
Programming techniques 764
Configuring logging properties for business objects 788
BusinessObject.properties file 791
Widget programming 792
Bindings 792
Export and import binding overview 794
Export and import binding configuration . . . 798
Interoperability between SCA modules and Open SCA services 809
Binding types 812
Developing business process management solutions 897
Business integration architecture and patterns 898

Chapter 7. Deploying modules directly to WebSphere ESB 903

Overview of preparing and installing modules . . 903
Libraries and JAR files overview 904
EAR file overview 906
Preparing to deploy to a server 906
Considerations for installing service applications on clusters 908
Deploying a mediation module 909
Installing versioned SCA modules in a production environment 910
Installing an SCA module with the console . . 911
Creating an installable EAR file using `serviceDeploy` 912
Deploying applications using Apache Ant tasks 913
Adapters and their installation 914

Chapter 8. Securing WebSphere ESB and applications 915

Getting started with security 916
Understanding elements of application security 916
Considerations for securing adapters 918
Creating end-to-end security 920
Defining RunAs roles user assignments for system applications 922
Configuring administrative and application security 922
Configuring administrative security 924
Configuring application security 927
Deploying (installing) secure applications 929
Assigning users to roles 930
Securing access to timetables in the Business Calendars widget 931
Roles associated with a timetable 931
System roles for the Security Roles widget . . 933
Assigning component roles 933
Setting up security for Business Space 934
Enabling security for Business Space 934
Selecting the user repository for Business Space 936
Setting up SSO and SSL for Business Space . . 940
Designating HTTP or HTTPS settings for Business Space 941
Setting up security for system REST services 941
Business Space widget security considerations 942

Configuring Tivoli Access Manager WebSEAL to work with Business Space 943
Assigning the Business Space superuser role 950
Assigning the Business Space superuser by user group 952
Preventing users from creating business spaces 954
Enabling searches for user registries without wildcards. 955

Chapter 9. Service component monitoring overview 959
Why use monitoring?. 959
What do you monitor? 960
How do you enable monitoring? 961

Chapter 10. Enabling and configuring service component monitoring 965
Monitoring performance. 965
Performance Monitoring Infrastructure statistics 966
Application Response Measurement statistics for the Service Component Architecture. 971
Monitoring service component events 984
Enabling monitoring of business process and human task events 985
Configuring logging for service component events 986
Monitoring service components with the Common Event Infrastructure server 991
Session monitoring 995

Chapter 11. Viewing monitored events 997
Viewing performance metrics with the Tivoli Performance Viewer 997
Viewing and interpreting service component event log files 998

Chapter 12. Event catalog 1001
The Common Base Event standard elements . . 1001
Business objects in events 1002
WebSphere Enterprise Service Bus events. . . . 1003
Resource Adapter events 1003
Binding events 1004
Business rule events. 1010
Business state machine events 1011
Map events. 1013
Mediation events. 1013
Recovery events 1014
Service Component Architecture events . . . 1014
Selector events 1015

Chapter 13. Samples. 1017
Running the StockQuote sample. 1018
Uninstalling the StockQuote sample 1020
List of samples in the online Samples Gallery 1021
The iteration aggregation sample 1021
The MQ data binding sample 1022
The XSLT mapping file migration sample . . 1023
The service gateway sample 1024
The Endpoint Lookup sample 1024

Chapter 14. Overview of troubleshooting 1027

Chapter 15. Troubleshooting checklist for WebSphere ESB 1031

Chapter 16. Messages overview . . . 1033

Chapter 17. WebSphere ESB log files 1035

Chapter 18. Transaction log file . . . 1037

Chapter 19. Troubleshooting installation and configuration 1039
Messages and known issues during installation and profile creation 1040
Informational messages 1040
Configuration errors were detected during the installation 1040
Supported IBM JDK was not found. The IBM JDK shipped with this product must be located at *install_root*/JDK. Correct this problem and try again. 1041
Warning: Cannot convert string “<type_name>” to type FontStruct 1041
CWWBB0627E error during installation with DB2 Express on Linux systems 1041
Installation and profile creation log files 1042
Launching Installation Manager directly on 64-bit systems 1044
Warnings about GTK or ulimit on Linux or UNIX when installing or migrating 1044
Troubleshooting the launchpad application or First Steps 1045
First Steps console fails to start on the Windows operating system 1046
Diagnosing a failing Ant configuration script . . 1046
Profile creation fails after reinstallation with DB2 Express 1048
Profile creation fails on Windows operating system 1048
Successful installation reported after profile creation failure 1049
Recovering from profile creation or augmentation failure 1049
Reinstallation cannot create new profile 1051

Chapter 20. Failure trying to serialize an object that is not serializable in a migrated BPEL file 1053

Chapter 21. Troubleshooting a failed deployment. 1055
Deleting JCA activation specifications 1056
Deleting SIBus destinations 1057

Chapter 22. Troubleshooting administration tasks and tools . . . 1059
Profile-specific log files 1059
Resolving browser display problems with administration tools 1062
Troubleshooting the failed event manager . . . 1062
Troubleshooting store-and-forward processing 1064

Chapter 23. Troubleshooting WebSphere Application Server . . . 1067

Chapter 24. Tools for troubleshooting your applications . . 1069
Debugging applications in IBM Integration Designer 1069
Using logging, tracing, and monitoring in your applications 1069
Collecting problem reports using IBM Support Assistant Lite 1069
Problem resolution using IBM Support Assistant Lite collectors 1070
Troubleshooting Service Component Architecture processing and call chains 1071
Managing failed events in WebSphere ESB . . . 1075
Security considerations for recovery 1078
Finding failed events in WebSphere ESB . . . 1078
Working with data in failed events 1081
Resubmitting failed events in WebSphere ESB 1085
Managing failed SCA events 1087
Managing failed JMS events 1088
Managing failed WebSphere MQ events . . . 1089
Finding Common Base Events related to a failed event 1091
Deleting failed events 1091
Troubleshooting the failed event manager . . 1092

Chapter 25. Recovering from a failure 1095
Overview of the recovery process 1095
Triggers for recovery 1095
Assessing the state of the system 1096
Recovery: Analyzing the problem 1098
Situational analysis 1099
Recovery: First steps 1099
Failed-event locations: Where does the data go? 1100
Use case: recovering data from failed events 1101
Recovery troubleshooting tips 1109
Restarting deployment environments 1109
Viewing the service integration bus. 1110
Capturing javacore 1113
Servers and recovery mode processing . . . 1115
Resolving indoubt transactions 1115
Reviewing DB2 diagnostic information . . . 1117
About recovering the messaging subsystem 1118
Using IBM Support Assistant 1119

Chapter 26. Searching knowledge bases 1123

Chapter 27. Using IBM Support Assistant 1125

Chapter 28. Getting fixes 1129

Chapter 29. Contacting IBM Software Support 1131

Chapter 30. Managing applications with WebSphere Application Server . 1133

Index 1135

Notices 1139
Programming interface information. 1141
Trademarks. 1141

Sending your comments to IBM . . . 1143

Figures

1. An enterprise service bus 5
2. Simplified example of a mediation module 12
3. Simplified example of an EAR file containing a mediation module 13
4. Mediation module containing three mediation primitives 14
5. Business Space and Business Process Management products 22
6. Detailed schematic of a WebSphere Adapter 24
7. Representing data, invocation, and composition within a programming model for SOA 38
8. Structure of a component 41
9. Service component structure 41
10. Mediation flow component between three service requesters or exports and two service providers or imports 43
11. Mediation flow between operations 43
12. Business objects represent the data that flows between services in an application 47
13. An export with HTTP binding 53
14. An import with HTTP binding 54
15. Flow of a request through the export to a component 55
16. Flow of a response back through the export 56
17. Flow from a component through the import to a service 57
18. Flow of a response back through the import 58
19. The function selector 63
20. How fault information is sent from the component through the export binding to the client. 66
21. Interface with two faults 66
22. How fault information is sent from the service through the import to the component 68
23. Component in SCA module invoking Open SCA service 70
24. Open SCA service invoking component in SCA module 71
25. How the Web service (JAX-WS) export binding processes a SOAP message with a referenced attachment 86
26. A SOAP message with an swaRef element 87
27. How the Web service (JAX-WS) export binding processes a SOAP message with an swaRef attachment 88
28. How swaRef attachments appear in the SMO 89
29. How the Web service (JAX-WS) import binding generates a SOAP message with an swaRef attachment 90
30. How an swaRef attachment in the SMO is accessed to create the SOAP message 90
31. How the Web service (JAX-WS) export binding processes a WS-I compliant SOAP message with a referenced attachment. 93
32. How referenced attachments appear in the SMO 93
33. How the referenced attachment in the SMO is accessed to create the SOAP message 95
34. A SOAP message with an unreferenced attachment 97
35. An attachment passing through an SCA module 97
36. A message processed by a mediation flow component 98
37. An attachment obtained from a database and added to the SOAP message 99
38. Flow of a request from the SCA application to the Web application 102
39. Flow of a request from the Web service to the client application. 102
40. Local invocation of an EJB (EJB 3.0 only) 109
41. Remote invocation of an EJB 110
42. Remote call from a client to an SCA component by way of an EJB export 111
43. Flow from a Siebel system to an SAP system 117
44. JMS import binding resources 124
45. JMS export binding resources 125
46. Simple use-case scenario: external client interacts with server application 129
47. Generic JMS import binding resources 133
48. Generic JMS export binding resources 134
49. WebSphere MQ JMS import binding resources 140
50. WebSphere MQ JMS export binding resources 141
51. WebSphere MQ import binding resources 147
52. WebSphere MQ export binding resources 148
53. Dynamic routing using the Endpoint Lookup mediation primitive 176
54. The SOA life cycle with security 183
55. IBM SOA security reference model 184
56. IT security services. 184
57. Security policy infrastructure 186
58. Transaction covering service requester, mediation flow and service provider 189
59. Simplified example of a message flowing through an ESB 194
60. Supported export bindings 195
61. Supported import bindings 195
62. Using an MQ binding. 197
63. Using an MQ JMS binding 198
64. Using a WebSphere JDBC adapter. 199
65. Using a JDBC adapter. 199
66. Using mediation primitives inside a mediation module 200
67. Using a JCA adapter 201
68. Using Web services 201
69. Using JMS 202
70. Using a CICS JCA adapter 203
71. Using the CICS Web service interface 203
72. Using the MQ-CICS bridge 204

© Copyright IBM Corp. 2007, 2011

73. Multiple access points compared to one access point 206
74. Overview of a proxy gateway request 211
75. Messages can be altered according to the context in which they occur. 213
76. At run time, the service provider receives a context-specific message 217
77. At run time, logging is done when a course is paid for 219
78. Services can be routed to different endpoints, depending on whether the requester has an associated SLA 220
79. How the SLA Check properties relate to the client and the final service provider. 222
80. How the SLA Endpoint Lookup properties relate to the client and the final service provider 224
81. How the SLA Endpoint Lookup properties relate to the client and the final service provider 226
82. Service connectivity scenario for stock quotes: overview 228
83. Service connectivity scenario for stock quotes: version 1.0 message flow overview 229
84. Mediation flow for stock quote version 1.0 230
85. Mediation flow for stock quote version 1.1: improved flexibility 231
86. Mediation flow for stock quote version 1.2: improved flexibility and reuse 232
87. Overview :Actor (service requester) and system (ESB). 234
88. Request flow for static service gateway 237
89. Response flow for static service gateway 238
90. Request flow for dynamic service gateway 240
91. Response Flow for dynamic gateway binding 240
92. Request flow for dynamic body-aware service gateway 242
93. Response flow for dynamic gateway binding 243
94. Multiprotocol service gateway 245
95. Actor (service requester) and system (ESB) 247
96. Dynamic Routing using the Endpoint Lookup mediation primitive 249
97. Actor (service requester) and system (ESB) 249
98. Dynamic Routing using the Database Lookup mediation primitive 251
99. Actor (service requester) and system (ESB) 251
100. Dynamic Routing using the Message Filter mediation primitive 253
101. Actor (service requester) and system (ESB) 253
102. The following figure shows a Dynamic Response Redirection when using the Database Lookup mediation primitive . . . 255
103. Actor (service requester) and system (ESB) 256
104. Controlling the behavior of service requests using mediation policies 259
105. Actor (service requester) and system (ESB) 259
106. Message protocol conversion using a mediation module 261
107. Actor (service requester) and system (ESB) 261
108. Message validation using the Database Lookup and Message Element Setter mediation primitives 263
109. Mediation policy document for groupA of SimpleApp 274
110. Properties for SCA module SimpleApp 274
111. Mediation policy for groupA of SimpleApp 276
112. Mediation policy attachment for SimpleApp 277
113. Gate condition on a mediation policy attachment 279
114. Classification URI 281
115. Governed mediation policy 282
116. Business Space Login 308
117. Administering mediation policies associated with modules, using the Mediation Policy Administration widget 309
118. Adding assertions, using the Mediation Policy Administration widget 311
119. Adding assertions and gate conditions, using the Mediation Policy Administration widget . 313
120. Selecting an existing mediation policy, to attach to a module. 316
121. Business Space Login 319
122. Administering mediation policies associated with services, using the Mediation Policy Administration widget 320
123. Selecting the group from which to create a mediation policy, using the Mediation Policy Administration widget 323
124. Adding assertions and gate conditions, using the Mediation Policy Administration widget . 326
125. Selecting an existing mediation policy, to attach to a target service 329
126. Business Space Login 332
127. A proxy gateway and a proxy group 333
128. Adding virtual services to proxy groups 335
129. Listing proxy gateways and proxy groups 336
130. Adding a Web service endpoint to a virtual service 337
131. WebSphere Enterprise Service Bus and WebSphere Service Registry and Repository interaction when creating an endpoint. . . . 341
132. The suggested end-to-end scenario for SLA endpoint resolution 353
133. A guided activity 360
134. Example showing one mediation module interacting with another mediation module . 413
135. Detailed schematic of a WebSphere Adapter 546
136. Detailed schematic of a WebSphere Business Integration Adapter. 547
137. Flow of asynchronous request processing from an SCA module to an target service when the store-and-forward feature is used . 583
138. Flow of asynchronous request processing from one SCA module to multiple target services when the store-and-forward feature is used. 584
139. Behavior of the servers in a cluster when the store-and-forward feature is activated . . . 586

140. Behavior of the servers in a cluster when the store-and-forward feature is started for one server in business space 587
141. Behavior of the servers in a cluster when the store is stopped in one server in business space 588
142. Shows the connection between the service requester, service provider and ESB 593
143. WebSphere ESB, installed on top of WebSphere Application Server 595
144. Simple service proxy pattern 596
145. Service selector pattern 597
146. Service translator pattern 598
147. Service gateway pattern 599
148. Message enrichment pattern 600
149. Batch aggregation pattern 601
150. Multiple source aggregation pattern 601
151. Structure of a module 603
152. Mediation module 604
153. Mediation flows 608
154. Overview of SMO structure 611
155. Mediation primitive 614
156. Branched mediation primitive 614
157. Message filter mediation primitive 615
158. Type Filter mediation primitive 616
159. Flow Order mediation primitive 617
160. Custom mediation primitive 617
161. Hover pane showing the source and target XML message type 619
162. Message Structure mapping using an XSLT mediation primitive within a mediation flow component 621
163. Message Structure mapping within Integration Designer 622
164. Service enrichment using a BO mapper within a mediation flow component 622
165. Format date function 623
166. The configuration of the format date transformation 623
167. Custom transformation in a BO map 623
168. Java code in the Properties view 623
169. Hover pane showing the message type and structure 625
170. Hoverpane showing the body structure of the SMO 626
171. Message Element Setter in a mediation flow component 627
172. Message Element Setter properties within Integration Designer 628
173. Data Handler Configuration 629
174. Data Handler within a mediation flow component 630
175. Data Handler in Integration Designer 630
176. WebSphere ESB shows how a component can be called synchronously or asynchronously . 634
177. Synchronous SCA invocation 635
178. Asynchronous with deferred response SCA invocation 637
179. Car insurance comparison website using asynchronous with deferred response invocation 637
180. Asynchronous with callback SCA invocation 639
181. Dynamic override of a fixed endpoint using an Endpoint Lookup mediation primitive . . 643
182. Dynamic invocation with a target import 644
183. Pure dynamic invocation. 645
184. Endpoint Lookup within a mediation flow 647
185. Endpoint Lookup mediation primitive properties 647
186. Try alternate endpoint property of the Callout node 647
187. SLA Endpoint Lookup 649
188. General pattern for a service gateway module 651
189. Five structures that represent raw data 652
190. DataHandler mediation primitive used in a service gateway 653
191. Automatic parsing used in a service gateway 654
192. Pattern Explorer in IBM Integration Designer 654
193. Pattern Explorer configuration view 655
194. Pattern Explorer summary information 656
195. Common pattern that is used for a dynamic service gateway 657
196. Static service gateway pattern 658
197. Proxy gateway 659
198. High level interactions of the proxy gateway, the configuration store and the web based interface 660
199. Proxy gateway widget in Business Space 662
200. Select a proxy gateway widget 663
201. Editing a new proxy group 663
202. Editing a new proxy group 664
203. The Fan Out mediation primitive properties view 666
204. The Fan In mediation primitive properties view 667
205. Example of an aggregation block 668
206. Shared context example 669
207. Aggregation Fan Out branching 671
208. Shared context response messages 671
209. Batch systems business objects 672
210. The Customer business object 672
211. Batch request structure 673
212. Batch response structure 673
213. DispatchOrder array 673
214. Fan Out/Fan In flow detail 674
215. Fan Out/Fan In properties view 674
216. Message Element Setter properties 674
217. Transform inline map 675
218. Message Element Setter aggregateResult properties 675
219. SMO mapping 675
220. Flow input and output 676
221. HR and Payroll Record 676
222. The inner and outer flow 677
223. EmployeeRecord array 677
224. PayslipStatus object 677
225. Business objects for batch order example 678
226. Result of batched order example 678
227. Aggregation levels 678
228. Shared outlet responses 679
229. Aggregation block 679
230. Branched aggregation block 680

231. Service invoke mediation primitive property details 680
232. Setting the Asynchronous invocation qualifier 681
233. Setting the number of requests that should be processed 681
234. Deleting the message body 682
235. Aggregation context mapping 683
236. The failInfo element in the message context 684
237. Runtime failure and rollback transaction 684
238. Example of a modelled fault 685
239. Stop mediation primitive. 686
240. An example error flow to return an unmodeled fault 687
241. An example error flow to return a modelled fault 688
242. Synchronous invocation error handling 688
243. Asynchronous invocation error handling 689
244. Asynchronous invocation error handling without queue 2 689
245. Asynchronous invocation error handling 690
246. Synchronous and asynchronous invocation error handling 690
247. Asynchronous and synchronous invocation error handling 691
248. Integration Test Client properties 692
249. Input parameters 692
250. Showing the relationship between a module in IBM Integration Designer and the module definition 699
251. Service component definition including component name, implementation, interfaces, and references 700
252. Example of a component definition in Service Component Definition Language 701
253. Example of an import definition in Service Component Definition Language 703
254. Example of an export definition in Service Component Definition Language 704
255. Clients can call a service component using either stand-alone or inline references . . . 705
256. Example of inline reference definitions 706
257. Assembly diagram 707
258. Example of an interface definition within a SCDL component definition. 710
259. Synchronous invocation 719
260. Asynchronous invocation models 720
261. Summary of invocation models along with supported methods for passing data 722
262. Invoking a service on a different system 727
263. Invoking a service on a different system 730
264. XML schema simple types 746
265. Comparison of flat and hierarchical business objects 749
266. Business graph complex type 752
267. An export with HTTP binding 793
268. An import with HTTP binding. 793
269. Flow of a request through the export to a component 795
270. Flow of a response back through the export 796
271. Flow from a component through the import to a service 797
272. Flow of a response back through the import 798
273. The function selector 803
274. How fault information is sent from the component through the export binding to the client 806
275. Interface with two faults 806
276. How fault information is sent from the service through the import to the component . 808
277. Component in SCA module invoking Open SCA service 810
278. Open SCA service invoking component in SCA module 811
279. How the Web service (JAX-WS) export binding processes a SOAP message with a referenced attachment. 826
280. A SOAP message with an swaRef element 827
281. How the Web service (JAX-WS) export binding processes a SOAP message with an swaRef attachment. 828
282. How swaRef attachments appear in the SMO 829
283. How the Web service (JAX-WS) import binding generates a SOAP message with an swaRef attachment. 830
284. How an swaRef attachment in the SMO is accessed to create the SOAP message . . . 830
285. How the Web service (JAX-WS) export binding processes a WS-I compliant SOAP message with a referenced attachment . . . 833
286. How referenced attachments appear in the SMO 833
287. How the referenced attachment in the SMO is accessed to create the SOAP message . . . 835
288. A SOAP message with an unreferenced attachment 837
289. An attachment passing through an SCA module 837
290. A message processed by a mediation flow component 838
291. An attachment obtained from a database and added to the SOAP message 839
292. Flow of a request from the SCA application to the Web application 842
293. Flow of a request from the Web service to the client application. 842
294. Local invocation of an EJB (EJB 3.0 only) 849
295. Remote invocation of an EJB 850
296. Remote call from a client to an SCA component by way of an EJB export 851
297. Flow from a Siebel system to an SAP system 857
298. JMS import binding resources 864
299. JMS export binding resources 865
300. Simple use-case scenario: external client interacts with server application 869
301. Generic JMS import binding resources 873
302. Generic JMS export binding resources 874
303. WebSphere MQ JMS import binding resources 880
304. WebSphere MQ JMS export binding resources 881
305. WebSphere MQ import binding resources 887
306. WebSphere MQ export binding resources 888
307. Relationships among module, component, and library 905

308. Example of roles assigned to a timetable 932
309. ARM statistics obtained from an SCA call with a synchronous implementation 974
310. Graphic representation of a deferred response with synchronous implementation 975
311. A deferred response with an asynchronous result retrieve 979
312. Diagram of an asynchronous callback with a synchronous implementation 980
313. An asynchronous callback with an asynchronous implementation 982
314. Diagram of asynchronous one-way call with a synchronous implementation 983
315. An asynchronous one-way call with an asynchronous implementation 984
316. StockQuote overview 1019
317. A remote messaging and remote support topology 1074
318. A remote messaging and remote support topology 1074
319. Assembly diagram of account routing process 1101
320. Assembly diagram of account routing - invocations 7 and 8 1103
321. Failed event manager processing. 1104

Tables

1. Icons that prefix links to topics outside this book . . . iii
2. Bidirectional language format string values 31
3. Predefined data handlers . . . 60
4. Predefined data bindings for JMS bindings 61
5. Predefined data bindings for WebSphere MQ bindings . . . 62
6. Predefined data bindings for HTTP bindings 62
7. Predefined function selectors for JMS bindings 63
8. Predefined function selectors for WebSphere MQ bindings . . . 64
9. Predefined function selectors for HTTP bindings . . . 64
10. Prepackaged fault selectors . . . 68
11. How security headers are passed . . . 77
12. How the attachment is generated . . . 96
13. How the attachment is generated . . . 96
14. Supplied HTTP header information . . . 104
15. Return values . . . 115
16. Typical stand-alone environment setup 167
17. Typical deployment environment setup 167
18. Supplied patterns and their relationship to features . . . 300
19. Default parameters and their assigned values 350
20. Graphical buttons at the top of a console collection page . . . 361
21. Buttons at the bottom of a console page 362
22. WebSphere ESB configuration files . . . 363
23. Deployment environment status . . . 386
24. Aggregated state of entities . . . 387
25. States of a topology instance in order of least to most available . . . 387
26. EJB import JNDI name configurations 431
27. Example values for import bindings . . . 435
28. Example values for export bindings . . . 435
29. Generic JMS imports: Names and JNDI names of resources created at installation on the server . . . 439
30. Generic JMS exports: Names and JNDI names of resources created at installation on the server . . . 439
31. MQ JMS imports: Names and JNDI names of resources created at installation on the server . 441
32. MQ JMS exports: Names and JNDI names of resources created at installation on the server . 441
33. Custom properties for WebSphere MQ queue destinations . . . 444
34. WebSphere MQ import: Names and JNDI names of resources created at installation on the server . . . 446
35. WebSphere MQ export: Names and JNDI names of resources created at installation on the server . . . 446
36. Custom properties for WebSphere MQ queue destinations . . . 448
37. Buttons for administering enterprise applications . . . 471
38. Icons in the Service Integration Bus Browser 480
39. Relationship database view columns . . . 501
40. Clarify customer . . . 502
41. SAP customer . . . 503
42. Siebel customer . . . 503
43. Business object definitions for customer on each database . . . 503
44. ID relationship definition . . . 503
45. RELN_VIEW_META_T table . . . 504
46. View column definition . . . 504
47. View column definition . . . 504
48. Logger names . . . 533
49. Differences between WebSphere Adapters and WebSphere Business Integration Adapters . . 547
50. Event sequencing support in a network deployment environment . . . 556
51. Sample output from esAdmin listLocks command . . . 561
52. Search criteria . . . 568
53. Failed SCA events . . . 575
54. Failed JMS events . . . 576
55. Failed WebSphere MQ events . . . 578
56. Behavior of the store-and-forward feature based on qualifier configuration . . . 590
57. Considerations for using the Service Invoke mediation primitive or the Callout node . . 632
58. Invocation style used by the mediation flow component for service invocations . . . 640
59. Which output terminal is fired from the Fan In mediation primitive . . . 670
60. Primary artifacts that make up an SCA service module . . . 697
61. Summary of key methods and interfaces for dynamic client invocation . . . 722
62. Summary of qualifiers . . . 732
63. WSDL type to Java class conversion . . . 737
64. Data abstractions and the corresponding implementations . . . 741
65. Business object services . . . 761
66. Logging specification descriptions . . . 789
67. Logging levels . . . 790
68. Scenarios where the VERSION parameter will cause behavior changes . . . 791
69. Predefined data handlers . . . 800
70. Predefined data bindings for JMS bindings 801
71. Predefined data bindings for WebSphere MQ bindings . . . 802
72. Predefined data bindings for HTTP bindings 802
73. Predefined function selectors for JMS bindings . . . 803
74. Predefined function selectors for WebSphere MQ bindings. . . . 804
75. Predefined function selectors for HTTP bindings . . . 804

© Copyright IBM Corp. 2007, 2011

76. Prepackaged fault selectors 808
77. How security headers are passed 817
78. How the attachment is generated 836
79. How the attachment is generated 836
80. Supplied HTTP header information 844
81. Return values 855
82. EAR files and associated roles in WebSphere ESB 917
83. Administrative security roles 926
84. PMI statistics for events 967
85. Event types and elements that can produce event performance statistics 967
86. Event types and elements that can produce ARM statistics 971
87. **Common** 972
88. **Reference** 972
89. **Target** 972
90. **Callback** 973
91. ARM statistics for synchronous invocations of SCA 974
92. Invocation of request and return result 975
93. Invocation of output to data source 975
94. Invocation of request and return result 976
95. Invocation of return result to a data store 977
96. Invocation of request and return result 978
97. Invocation of request and return result 979
98. Invocation of request and return result 980
99. Invocation of callback 980
100. Invocation of request and return result 981
101. Invocation of callback 982
102. Invocation of request and return result 983
103. Invocation of request and return result 984
104. JMS binding 1005
105. JAX-WS binding 1007
106. HTTP binding events 1008
107. EJB binding events 1009
108. EIS binding events 1009
109. Base element 1013
110. Installation and profile logs for WebSphere ESB components 1043
111. Profile-specific log files updated during runtime 1059
112. Search criteria 1080
113. Failed SCA events 1087
114. Failed JMS events 1088
115. Failed WebSphere MQ events 1090
116. Invocation patterns and relationship to the creation of failed events: **Service Business Exceptions** 1105
117. Invocation patterns and relationship to the creation of failed events: **Service Runtime Exceptions** 1105
118. Administrative capabilities to help manage failures 1108

Chapter 1. Product overview for WebSphere ESB

WebSphere® ESB (WebSphere ESB) supports the integration of service-oriented, message-oriented, and event-driven technologies to provide a standards-based messaging infrastructure to companies wanting a fast start to an Enterprise Service Bus.

Related information:

PDF documentation
WebSphere Enterprise Service Bus documentation (in PDF format)

Information roadmaps
Business Process Management information roadmaps on IBM developerWorks organize information about WebSphere ESB and the other products in the portfolio.

IBM Education Assistant
Multimedia educational modules about WebSphere ESB, provided by IBM® Education Assistant.

Overview
Overview tab, on product library Web page. Use this page to access announcements, data sheets, and other general library documents related to WebSphere ESB.

WebSphere ESB and the enterprise service bus

WebSphere ESB provides the capabilities of a standards-based enterprise service bus.

WebSphere ESB manages the flow of messages between service requesters and service providers. *Mediation modules* within the ESB handle mismatches between requesters and providers, including protocol or interaction-style, interface and quality of service mismatches. In an SCA-based solution mediation modules are a type of SCA module. The mediation modules perform a special role, and therefore have slightly different characteristics from other components that operate at the business level.

Mediation components operate on messages exchanged between service endpoints. In contrast with regular business application components, they are concerned with the flow of the messages through the infrastructure and not just with the business content of the messages. Rather than performing business functions, they perform routing, transformation, and logging operations on the messages. The information that governs their behavior is often held in headers flowing with the business messages. The IBM SOA programming model introduces the *service message object* (SMO) data structure to support this pattern.

WebSphere ESB supports advanced interactions between service endpoints on three levels: broad connectivity, a spectrum of interaction models and qualities of interaction, and mediation capabilities. The product supports connectivity between endpoints through a variety of protocols and application programming interfaces (APIs):

- Java Message Service (JMS) 1.1. Applications can exploit a variety of transports, including TCP/IP, SSL, HTTP, and HTTPS.

© Copyright IBM Corp. 2007, 2011

- WebSphere ESB supports Web services standards that enable applications to make use of Web service capabilities.
  - SOAP/HTTP, SOAP/JMS, WSDL 1.1
  - UDDI 3.0 Service Registry, through the WebSphere Integration Developer
  - WS-* Standards including WS-Security, WS-Atomic Transactions

Because it is built on WebSphere Application Server, WebSphere ESB can provide smooth interoperability with other products in the WebSphere portfolio, including IBM WebSphere MQ and IBM WebSphere Message Broker. It can also, with IBM WebSphere Adapters solutions, use existing application assets, as well as capture and disseminate business events.

The message clients for C/C++ and Microsoft .NET enable non-Java applications to connect to WebSphere ESB using an API similar to the JMS API.

Other features at the connectivity level perform basic protocol conversion between endpoints where the protocol used by the requester to dispatch requests (such as SOAP over HTTP) is different from that of the service provider that is to handle those requests (such as SOAP over JMS).

What's new in this release

WebSphere ESB Version 7.5.1 provides enhanced mediation flow processing and a new mediation primitive.

Note: The information center has been updated for IBM WebSphere ESB version 7.5.1. For more information about installing fix packs, see Getting fixes.

WebSphere ESB, version 7.5.1 includes the following new features:

- Enhanced developer productivity:
  - A WebSphere environment variable is available to enable mediation flows and other resources to be loaded at application startup, instead of during the first invocation. The result is that the specified mediation flows are ready for processing when the first message arrives.
  - The Synchronous Transaction Rollback mediation primitive is available to roll back the current transaction and then continue processing the flow. The Synchronous Transaction Rollback can only be used in mediation flows that use a synchronous invocation style.
  - Development Guide explains the programming model and guides feature usage.

Release notes

You can view the latest WebSphere ESB version 7.5.1 technotes at:
http://www.ibm.com/support/search.wss?q=websphere+enterprise+service+bus

For more general support information for WebSphere ESB see the support pages at http://www.ibm.com/software/integration/wsesb/support/.

Architectural overview of WebSphere ESB

WebSphere ESB is a service-oriented architecture (SOA) integration platform built on a uniform invocation programming model and a uniform data representation model.

The base runtime infrastructure for WebSphere ESB is WebSphere Application Server. The Service Component Architecture and business objects that are part of the SOA core provide the uniform invocation and data-representation programming models. The SOA core includes the Common Event Infrastructure for generating events for the monitoring and management of applications running on WebSphere ESB.

The combination of a powerful foundation (WebSphere Application Server and the SOA Core) and service components in WebSphere ESB allows quick development and deployment of sophisticated composite applications that run on WebSphere ESB.

Service-oriented architecture core

The service-oriented architecture core of IBM WebSphere ESB provides both uniform invocation and data-representation programming models and monitoring and management capabilities for applications running on WebSphere ESB.

Service-oriented architecture (SOA) is a conceptual description of the structure of a software system in terms of its components and the services they provide, without regard for the underlying implementation of these components, services and connections between components. WebSphere ESB enables deployment of standards-based process integration solutions in an SOA. This means that a well defined set of business-level interfaces for the components can be created and maintained, shielded from lower-level technology changes. Loosely coupled integration applications that are based on SOA provide flexibility and agility. You can implement integration solutions independent of platform, protocols and products. For more information about SOA, refer to the Service-Oriented Architecture (SOA) from IBM Web site.

The Service Component Architecture and business objects that are part of the SOA core provide uniform invocation and data-representation programming models for applications deployed on WebSphere ESB. The SOA core also includes the Common Event Infrastructure for generating events for the monitoring and management of applications on WebSphere ESB.

Service Component Architecture

Service Component Architecture presents all elements of business transactions in a service-oriented way in the WebSphere ESB runtime environment.

Service Component Architecture (SCA) is an architecture in which all elements of a business transaction, such as access to web services, enterprise information system (EIS) service assets, business rules, workflows, databases and so on, are represented in a service-oriented way.

SCA separates business logic from implementation, so that you can focus on assembling an integrated application without knowing implementation details. The implementation of BPEL processes is contained in service components.

Service components can be assembled graphically in IBM Integration Designer, and the implementation can be added later. The SCA programming model narrows what developers must know about Java and Java EE or other implementation in particular scenarios to a core set of language concepts that are familiar to all who develop business applications in other programming languages today. This allows developers to quickly and easily integrate technologies.

Developers switching from classical application development environments face a much smaller learning curve; they can quickly become productive with this programming model. The Service Component Architecture programming model also helps experienced Java EE developers be more productive.

Service qualifiers govern the interaction between a service client and a service on the WebSphere ESB runtime environment. Service qualifiers are quality of service specifications that define a set of communication characteristics required by an application for transmission priority, level of route reliability, transaction management, and security level. An application communicates its quality of service needs to a runtime environment by specifying service qualifiers. You can specify service qualifiers when wiring components in the assembly editor in IBM Integration Designer. These specifications, when running on WebSphere ESB, determine how the clients interact with the target components. Depending on the qualifiers specified, additional required processing can take place at run time.

Importing and exporting capabilities within the Service Component Architecture define a service module's external interfaces or access points for WebSphere ESB. Imports and exports can be either to other modules within the same application, or to other applications on enterprise information systems (EIS). This allows working with IBM WebSphere Adapters. For more information about imports and exports, see Service applications.

WebSphere ESB solutions rely upon the underlying WebSphere Application Server capabilities for transaction, security, and workload management to provide a scalable integration environment.

Connecting services through an enterprise service bus

With an enterprise service bus (ESB), you can maximize the flexibility of an SOA. Participants in a service interaction are connected to the ESB, rather than directly to one another.

When the service requester connects to the ESB, the ESB takes responsibility for delivering its requests, using messages, to a service provider offering the required function and quality of service. The ESB facilitates requester-provider interactions and addresses mismatched protocols, interaction patterns, or service capabilities. An ESB can also enable or enhance monitoring and management. The ESB provides virtualization and management features that implement and extend the core capabilities of SOA.

The ESB abstracts the following features:

Location and identity
: Participants need not know the location or identity of other participants. For example, requesters need not be aware that a request could be serviced by any of several providers; service providers can be added or removed without disruption.

Interaction protocol
: Participants need not share the same communication protocol or interaction style. For example, a request expressed as SOAP over HTTP can be serviced by a provider that only understands SOAP over Java Message Service (JMS).

Interface
: Requesters and providers need not agree on a common interface. An ESB reconciles differences by transforming request and response messages into a form expected by the provider.

Requesters and providers need not agree on a common interface
An ESB reconciles differences by transforming request messages into a form expected by the provider.

Qualities of (interaction) service
Participants, or systems administrators, declare their quality-of-service requirements, including authorization of requests, encryption and decryption of message contents, automatic auditing of service interactions, and how their requests should be routed (for example, optimizing for speed or cost).

Interposing the ESB between participants enables you to modulate their interaction through a logical construct called a *mediation*. Mediations operate on messages in-flight between requesters and providers. For example, mediations can be used to find services with specific characteristics that a requester is asking for, or to resolve interface differences between requesters and providers. For complex interactions, mediations can be chained sequentially.

An enterprise service bus, with mediations, performs the following actions between requester and service:

- *Routing* messages between services. An enterprise service bus offers a common communication infrastructure that can be used to connect services, and thereby the business functions they represent, without the need for programmers to write and maintain complex connectivity logic.
- *Converting* transport protocols between requester and service. An enterprise service bus provides a consistent, standards-based way to integrate business functions that use different IT standards. This enables integration of business functions that could not normally communicate, such as to connect applications in departmental silos or to enable applications in different companies to participate in service interactions.
- *Transforming* message formats between requester and service. An enterprise service bus enables business functions to exchange information in different formats, with the bus ensuring that the information delivered to a business function is in the format required by that application.
- *Handling* business events from disparate sources. An enterprise service bus supports event-based interactions in addition to the message exchanges to handle service requests.

Figure 1. An enterprise service bus. The enterprise service bus is routing messages between applications, which are requesters or providers of services. The bus is converting transport protocols and transforming message formats between requesters and providers. In this figure, each application uses a different protocol (represented by the different geometric shapes of their connectors) and uses different message formats.

By using the enterprise service bus you can concentrate focus on your core business rather than your computer systems. You can change or add to the services when you need to; for example, to respond to changes in the business requirement,

to add extra service capacity, or to add new capabilities. You can make the required changes by reconfiguring the bus, with little or no impact to existing services and applications that use the bus.

The business object framework

Business objects define the data flowing between components defined in Service Component Architecture.

Part of the IBM WebSphere Application Server capabilities that are built into WebSphere ESB, business objects provide a framework for data application development that simplifies the Java EE data programming model.

The business object framework, included in WebSphere ESB as part of the Service-oriented architecture (SOA) core, provides a universal means of describing and exchanging data between Service Component Architecture services – for example, JDBC ResultSet and XML Schema described data.

A *business object* is a set of attributes that represent a business entity (such as Employee), an action on the data (such as a create or update operation), and instructions for processing the data. Components of the integration application use business objects to exchange information and trigger actions. Business objects are flexible because they can represent any kind of data. For example, in addition to supporting the data canonicalization model of traditional integration servers, they also can represent data returned from a synchronous EJB Session Bean facade or a synchronous business process, and then they can be bound to IBM WebSphere Portal portlets and JSF components.

Business objects are the primary mechanism for representing business entities, or documenting literal message definitions, enabling everything from a simple basic object with scalar properties to a large, complex hierarchy or graph of objects.

In WebSphere ESB, the business object framework is made up of the following elements:

- Business object definition
- Business graph definition
- Business object metadata definition
- Business object services (service APIs)

A business object definition is the name, set of ordered attributes, properties, version number, and application-specific text that specify a type of business object. A business graph definition is the wrapper added around a simple business object or a hierarchy of business objects to provide additional capabilities, such as carrying change summary and event summary information related to the business objects in the business graph. A business object metadata definition is the metadata that can be added to business object definitions to enhance their value when running on WebSphere ESB. This metadata is added to the business object's XML schema definition as well known xs:annotation and xs:appinfo elements. Business object services are a set of capabilities provided on top of the basic capabilities provided by Service Data Objects. Examples are services such as create, copy, equality, and serialization.

For more information about WebSphere Application Server Service Data Objects, see the WebSphere Application Server Network Deployment documentation.

Common Event Infrastructure in WebSphere ESB

The Common Event Infrastructure is an embedded technology within WebSphere ESB to provide basic event management services.

The infrastructure portion of the Common Event Infrastructure is included as part of the underlying IBM WebSphere Application Server capabilities in WebSphere ESB. The event emitting capabilities are additional functions of WebSphere ESB.

The *Common Event Infrastructure (CEI)* is the implementation of a set of APIs and infrastructure for the creation, transmission, persistence, and distribution of business, system, and network Common Base Events. A *Common Base Event* is a specification based on XML that defines a mechanism for managing events – such as logging, tracing, management, and business events – in business enterprise applications.

CEI provides basic event-management services, including consolidating and persisting raw events from multiple, heterogeneous sources and distributing those events to event consumers. It provides functionality for generation, propagation, persistence, and consumption of events representing service component processes. A standard, XML-based format, the Common Base Event model, defines the structure of these events. Each type of event used by the server contains a number of standard fields specific to a given type of event. In some cases, it contains an encapsulation of the business object data that is being used by the service component at a particular event point.

WebSphere ESB uses events in the CEI almost exclusively to enable service component monitoring. You must configure the CEI server if you want to use event-related functions, but after that, you should not use CEI directly. Instead, use the existing services in WebSphere ESB.

In WebSphere ESB, a specially configured CEI Server -- which can be part of an existing process server or another server -- is used for all event-related services. You must first create and deploy several facilities that are used by the CEI Server, including an event database, a messaging engine, one or more enterprise applications, and a database driver.

Related information:

Administering the Common Event Infrastructure

Service applications and service modules

A service module is a Service Component Architecture (SCA) module that provides services in the run time. When you deploy a service module toWebSphere ESB, you build an associated service application that is packaged as an Enterprise ARchive (EAR) file.

Deploying a service application

The process of deploying an EAR file containing a service application is the same as the process of deploying any EAR file. You can modify values for mediation parameters at deployment time. After you have deployed an EAR file containing an SCA module, you can view details about the service application and its associated module. You can see how a service module is connected to service requesters (through exports) and service providers (through imports).

Viewing SCA module details

The service module details that you can view depend upon the SCA module. They can include the following attributes.

- SCA module name
- SCA module description
- Associated application name
- SCA module version information, if the module is versioned
- SCA module imports:
  - Import interfaces are abstract definitions that describe how an SCA module accesses a service.
  - Import bindings are concrete definitions that specify the physical mechanism by which an SCA module accesses a service. For example, using SOAP/HTTP.
- SCA module exports:
  - Export interfaces are abstract definitions that describe how service requesters access an SCA module.
  - Export bindings are concrete definitions that specify the physical mechanism by which a service requester accesses an SCA module, and indirectly, a service.
- SCA module properties

Imports and import bindings

Imports define interactions between SCA modules and service providers. SCA modules use imports to permit components to access external services (services that are outside the SCA module) using a local representation. Import bindings define the specific way that an external service is accessed.

If SCA modules do not need to access external services, they are not required to have imports. Mediation modules usually have one or more imports that are used to pass messages or requests on to their intended targets.

Interfaces and bindings

An SCA module import needs at least one interface, and an SCA module import has a single binding.

- Import interfaces are abstract definitions that define a set of operations using Web Services Description Language (WSDL), an XML language for describing Web services. An SCA module can have many import interfaces.
- Import bindings are concrete definitions that specify the physical mechanism that SCA modules use to access an external service.

Supported import bindings

WebSphere ESB supports the following import bindings:

- SCA bindings connect SCA modules to other SCA modules. SCA bindings are also referred to as default bindings.
- Web Service bindings permit components to invoke Web services. The supported protocols are SOAP1.1/HTTP, SOAP1.2/HTTP, and SOAP1.1/JMS.

  You can use a SOAP1.1/HTTP or SOAP1.2/HTTP binding based on the Java API for XML Web Services (JAX-WS), which allows interaction with services using document or RPC literal bindings and which uses JAX-WS handlers to

customize invocations. A separate SOAP1.1/HTTP binding is provided to allow interaction with services that use an RPC-encoded binding or where there is a requirement to use JAX-RPC handlers to customize invocations.

- HTTP bindings permit you to access applications using the HTTP protocol.
- Enterprise JavaBeans (EJB) import bindings enable SCA components to invoke services provided by Java EE business logic running on a Java EE server.
- Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved through the use of resource adapters.
- Java Message Service (JMS) 1.1 bindings permit interoperability with the WebSphere Application Server default messaging provider. JMS can exploit various transport types, including TCP/IP and HTTP or HTTPS. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported.
- Generic JMS bindings permit interoperability with third-party JMS providers that integrate with the WebSphere Application Server using the JMS Application Server Facility (ASF).
- WebSphere MQ JMS bindings permit interoperability with WebSphere MQ-based JMS providers. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported. If you want to use WebSphere MQ as a JMS provider, use WebSphere MQ JMS bindings.
- WebSphere MQ bindings permit interoperability with WebSphere MQ. You can use WebSphere MQ bindings only with remote queue managers by way of a WebSphere MQ client connection; you cannot use them with local queue managers. Use WebSphere MQ bindings if you want to communicate with native WebSphere MQ applications.

Dynamic invocation of services

Services can be invoked through any supported import binding. A service is normally found at an endpoint specified in the import. This endpoint is called a static endpoint. It is possible to invoke a different service by overriding the static endpoint. Dynamic override of static endpoints lets you invoke a service at another endpoint, through any supported import binding. Dynamic invocation of services also permits you to invoke a service where the supported import binding does not have a static endpoint.

An import with an associated binding is used to specify the protocol and its configuration for dynamic invocation. The import used for the dynamic invocation can be wired to the calling component, or it can be dynamically selected at runtime.

For Web service and SCA invocations, it is also possible to make a dynamic invocation without an import, with the protocol and configuration deduced from the endpoint URL. The invocation target type is identified from the endpoint URL. If an import is used, the URL must be compatible with the protocol of the import binding.

- An SCA URL indicates invocation of another SCA module.
- An HTTP or a JMS URL by default indicates invocation of a Web service; for these URLs, it is possible to provide an additional binding type value that indicates that the URL represents an invocation by way of an HTTP or JMS binding.

- For a Web service HTTP URL, the default is to use SOAP 1.1, and a binding type value can be specified that indicates the use of SOAP 1.2.

Exports and export bindings

Exports define interactions between SCA modules and service requesters. SCA modules use exports to offer services to others. Export bindings define the specific way that an SCA module is accessed by service requesters.

Interfaces and bindings

An SCA module export needs at least one interface.

- Export interfaces are abstract definitions that define a set of operations using Web Services Description Language (WSDL), an XML language for describing Web services. An SCA module can have many export interfaces.
- Export bindings are concrete definitions that specify the physical mechanism that service requesters use to access a service. Usually, an SCA module export has one binding specified. An export with no binding specified is interpreted by the run time as an export with an SCA binding.

Supported export bindings

WebSphere ESB supports the following export bindings:

- SCA bindings connect SCA modules to other SCA modules. SCA bindings are also referred to as default bindings.
- Web Service bindings permit exports to be invoked as Web services. The supported protocols are SOAP1.1/HTTP, SOAP1.2/HTTP, and SOAP1.1/JMS.

  You can use a SOAP1.1/HTTP or SOAP1.2/HTTP binding based on the Java API for XML Web Services (JAX-WS), which allows interaction with services using document or RPC literal bindings and which uses JAX-WS handlers to customize invocations. A separate SOAP1.1/HTTP binding is provided to allow interaction with services that use an RPC-encoded binding or where there is a requirement to use JAX-RPC handlers to customize invocations.
- HTTP bindings permit exports to be accessed using the HTTP protocol.
- Enterprise JavaBeans (EJB) export bindings allow SCA components to be exposed as EJBs so that Java EE business logic can invoke SCA components otherwise unavailable to them.
- Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved through the use of resource adapters.
- Java Message Service (JMS) 1.1 bindings permit interoperability with the WebSphere Application Server default messaging provider. JMS can exploit various transport types, including TCP/IP and HTTP or HTTPS. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported.
- Generic JMS bindings permit interoperability with third-party JMS providers that integrate with the WebSphere Application Server using the JMS Application Server Facility (ASF).
- WebSphere MQ JMS bindings permit interoperability with WebSphere MQ-based JMS providers. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported. If you want to use WebSphere MQ as a JMS provider, use WebSphere MQ JMS bindings.

- WebSphere MQ bindings permit interoperability with WebSphere MQ. You use a remote (or client) connection to connect to an MQ queue manager on a remote machine. A local (or bindings) connection is a direct connection to WebSphere MQ. This can be used only for a connection to an MQ queue manager on the same machine. WebSphere MQ will permit both types of connection, but MQ bindings only support the "remote" (or "client") connection.

Mediation modules

Mediation modules are Service Component Architecture (SCA) modules that can change the format, content, or target of service requests.

Mediation modules operate on messages that are in-flight between service requesters and service providers. You are able to route messages to different service providers and to amend message content or form. mediation modules can provide functions such as message logging, and error processing that is tailored to your requirements.

You can change certain aspects of mediation modules, from the administrative console, without having to redeploy the module.

Components of mediation modules

Mediation modules contain the following items:

- Imports, which define interactions between SCA modules and service providers. They allow SCA modules to call external services as if they were local. You can view mediation module imports and modify the binding.
- Exports, which define interactions between SCA modules and service requesters. They allow an SCA module to offer a service and define the external interfaces (access points) of an SCA module. You can view mediation module exports.
- SCA components, which are building blocks for SCA modules such as mediation modules. You can create and customize SCA modules and components graphically, using Integration Designer. After you deploy a mediation module you can customize certain aspects of it from the administrative console, without having to redeploy the module.

  Usually, mediation modules contain a specific type of SCA component called a *mediation flow component*. Mediation flow components define mediation flows.

  A mediation flow component can contain none, one, or a number of mediation primitives. WebSphere ESB supports a supplied set of mediation primitives that provide functionality for message routing and transformation. If you need additional mediation primitive flexibility, you can use the Custom Mediation primitive to call custom logic.

  The purpose of a mediation module that does not contain a mediation flow component is to transform service requests from one protocol to another. For example, a service request might be made using SOAP/JMS but might need transforming to SOAP/HTTP before sending on.

  Note: You can view and make certain changes to mediation modules from WebSphere ESB. However, you cannot view or change the SCA components inside a module fromWebSphere ESB. Use Integration Designer to customize SCA components.

Figure 2. Simplified example of a mediation module. The mediation module contains one mediation flow component, which contains mediation primitives.

- Properties

  Mediation primitives have properties, some of which can be displayed in the administrative console as additional properties of an SCA module.

  For mediation primitive properties to be visible from the WebSphere ESB administrative console, the integration developer must promote the properties. Certain properties lend themselves to being administratively configured and Integration Designer describes these as promotable properties, because they can be promoted from the integration cycle to the administrative cycle. Other properties are not suitable for administrative configuration, because modifying them can affect the mediation flow in such a way that the mediation module needs to be redeployed. Integration Designer lists the properties that you can choose to promote under the promoted properties of a mediation primitive.

  You can use the WebSphere ESB administrative console to change the value of promoted properties without having to redeploy a mediation module, or restart the server or module.

  Generally, mediation flows use property changes immediately. However, if property changes occur in a deployment manager cell, they take effect on each node as that node is synchronized. Also, mediation flows that are in-flight continue to use previous values.

  Note: From the administrative console, you can only change property values, not property groups, names or types. If you want to change property groups, names or types, you must use Integration Designer.
- A mediation module or dependent library may also define subflows. A subflow encapsulates a set of mediation primitives wire together as a reusable piece of integration logic. A primitive can be added to a mediation flow to invoke a subflow.

Deploying mediation modules

Mediation modules are created using Integration Designer, and are generally deployed to WebSphere ESB inside an enterprise archive (EAR) file.

You can change the value of promoted properties at deployment time.

You can export a mediation module from Integration Designer, and cause Integration Designer to package the mediation module inside a Java archive (JAR) file, and the JAR file inside an EAR file. You can then deploy the EAR file, by installing a new application from the administrative console.

Mediation modules can be thought of as one entity. However, SCA modules are defined by a number of XML files stored in a JAR file.

Figure 3. Simplified example of an EAR file containing a mediation module. The EAR file contains JARs. The utility JAR file contains a mediation module.

Mediation primitives

Mediation flow components operate on message flows between service components. The capabilities of a mediation component are implemented by *mediation primitives*, which implement standard service implementation types.

A mediation flow component has one or more flows. For example, one for request and one for reply.

WebSphere ESB supports a supplied set of mediation primitives, which implement standard mediation capabilities for mediation modules deployed into WebSphere ESB. If you need special mediation capabilities, you can develop your own custom mediation primitives.

A mediation primitive defines an “in” operation that processes or handles messages that are represented by service message objects (SMOs). A mediation primitive can also define “out” operations that send messages to another component or module.

Figure 4. Mediation module containing three mediation primitives

You can use Integration Designer to configure mediation primitives and set their properties. Some of these properties can be made visible to the runtime administrator by promoting them. Any mediation primitive property that can be promoted can also be a dynamic property. A dynamic property can be overridden, at run time, using a policy file.

Integration Designer also allows you to graphically model and assemble mediation flow components from mediation primitives, and assemble mediation modules from mediation flow components. The administrative console refers to mediation modules as SCA modules.

Integration Designer also allows the definition of subflows in modules or their dependent libraries. A subflow can contain any mediation primitive except for the Policy Resolution mediation primitive. A subflow is invoked from a request or response flow, or from another subflow using the Subflow mediation primitive. Properties promoted from mediation primitives in a subflow are exposed as properties on the Subflow mediation primitives. These may then be promoted again until they reach the module level at which point they can then be modified by the runtime administrator.

Supported mediation primitives

The following set of mediation primitives are supported by WebSphere ESB:

Business Object Map
Transforms messages.

- Defines message transformations using a business object map, which can be reused.
- Allows you to define message transformations graphically, using the business object map editor.
- Can alter the content of a message.
- Can transform an input message type to a different output message type.

Custom Mediation
Allows you to implement your own mediation logic in Java code. The Custom Mediation primitive combines the flexibility of a user-defined

mediation primitive, with the simplicity of a pre-defined mediation primitive. You can create complex transformations and routing patterns by:

- Creating Java code.
- Creating your own properties.
- Adding new terminals.

You can call a service from a Custom Mediation primitive, but the Service Invoke mediation primitive is designed to call services and provides additional functionality, such as retry.

Data Handler

Allows you to transform a part of a message. It is used to convert an element of a message from a physical format to a logical structure or a logical structure to a physical format. The primary usage of the primitive is to convert a physical format, such as a Text string within a JMS Text Message object, into a logical Business Object structure and back again. This mediation is commonly used to:

- Transform a section of the input message from a defined structure to another - an example of this would be were the SMO includes a string value that is comma delimited and you want to parse this into a specific Business Object.
- Alter the message type – an example would be when a JMS export has been configured to use a JMS basic typed data binding and within the mediation module the integration developer decides that the content should be inflated to a specific BO structure.

Database Lookup

Modifies messages, using information from a user-supplied database.

- You must set up a database, data source, and any server authentication settings for the Database Lookup mediation primitive to use. Use the administrative console to help you do this.
- The Database Lookup mediation primitive can read from only one table.
- The specified key column must contain a unique value.
- The data in the value columns must be either a simple XML schema type, or an XML schema type that extends a simple XML schema type.

Endpoint Lookup

Allows for the dynamic routing of requests, by searching for service endpoints in a repository.

- Service endpoint information is retrieved from a WebSphere Service Registry and Repository (WSRR). The WSRR registry can be local or remote.
- You make registry changes from the WSRR administrative console.
- WebSphere ESB needs to know which registry to use, therefore, you must create WSRR access definitions using the WebSphere ESB administrative console.

Event Emitter

Enhances monitoring by letting you send events from inside a mediation flow component.

- You can suspend the mediate action by deselecting the check box.
- You can view Event Emitter events using the Common Base Events browser on WebSphere ESB.
- You should only send events at a significant point in a mediation flow, for performance reasons.

- You can define the parts of the message that the event contains.
- The events are sent in the form of Common Base Events and are sent to a Common Event Infrastructure server.
- To fully use the Event Emitter information, event consumers need to understand the structure of the Common Base Events. The Common Base Events has an overall schema, but this does not model the application specific data, which is contained in the extended data elements. To model the extended data elements, the Integration Designer tools generate a Common Event Infrastructure event catalog definition file for each of the configured Event Emitter mediation primitives. Event catalog definition files are export artifacts that are provided to help you; they are not used by Integration Designer or by the WebSphere ESB runtime. You should refer to the event catalog definition files when you create applications to consume Event Emitter events.
- You can specify other monitoring from WebSphere ESB. For example, you can monitor events to be emitted from imports and exports.

Fail Stops a particular path in the flow, and generates an exception.

Fan In Helps aggregate (combine) messages.

- Can only be used in combination with the Fan Out mediation primitive.
- Together, the Fan Out and Fan In mediation primitives allow aggregation of data into one output message.
- The Fan In mediation primitive receives messages until a decision point is reached, then one message is output.
- The shared context should be used to hold aggregation data.

Fan Out

Helps split and aggregate (combine) messages.

- Together, the Fan Out and Fan In mediation primitives allow aggregation of data into one output message.
- In iterate mode, the Fan Out mediation primitive lets you iterate through a single input message that contains a repeating element. For each occurrence of the repeating element, a message is sent.
- The shared context should be used to hold aggregation data.

HTTP Header Setter

Provides a mechanism for managing headers in HTTP messages.

- Can create, set, copy, or delete HTTP message headers.
- Can set multiple actions to change multiple HTTP headers.

MQ Header Setter

Provides a mechanism for managing headers in MQ messages.

- Can create, set, copy, or delete MQ message headers.
- Can set multiple actions to change multiple MQ headers.

SOAP Header Setter

Provides a mechanism for managing headers in SOAP messages.

- Can create, set, copy, or delete SOAP message headers.
- Can set multiple actions to change multiple SOAP headers.

Message Element Setter

Provides a simple mechanism for setting the content of messages.

- Can change, add or delete message elements.
- Does not change the type of the message.

- The data in the value columns must be either a simple XML schema type, or an XML schema type that extends a simple XML schema type.

Message Filter
Routes messages down different paths, based on the message content.

- You can suspend the mediate action by deselecting the check box.

Message Logger
Logs messages in a relational database or through your own custom logger. The messages are stored as XML, therefore, data can be post-processed by XML-aware applications.

- You can suspend the mediate action by deselecting the check box.
- The rational database schema (table structure) is defined by IBM.
- By default, the Message Logger mediation primitive uses the Common database. The runtime maps the data source at `jdbc/mediation/messageLog` to the Common database.
- You can set Handler implementation classes to customize the behavior of the custom logger. Optionally, you can provide Formatter implementation classes, Filter implementation classes, or both to customize the behavior of the custom logger.

Policy Resolution
Allows for the dynamic configuration of requests, by searching for service endpoints, and associated policy files, in a repository.

- You can use a policy file to dynamically override the promoted properties of other mediation primitives.
- Service endpoint information and policy information is retrieved from a WebSphere Service Registry and Repository (WSRR). The WSRR registry can be local or remote.
- You make registry changes from the WSRR administrative console.
- WebSphere ESB needs to know which registry to use, therefore, you must create WSRR access definitions using the WebSphere ESB administrative console.

Service Invoke
Calls a service from inside a mediation flow, rather than waiting until the end of the mediation flow and using the callout mechanism.

- If the service returns a fault, you can retry the same service or call another service.
- The Service Invoke mediation primitive is a powerful mediation primitive that can be used on its own for simple service calls, or in combination with other mediation primitives for complex mediations.

Set Message Type
During integration development, lets you treat weakly-typed message fields as though they are strongly-typed. A field is weakly-typed if it can contain more than one type of data. A field is strongly-typed if its type and internal structure are known.

- At runtime, the Set Message Type mediation primitive lets you check that the content of a message matches the data types you expect.

Stop Stops a particular path in the flow, without generating an exception.

Type Filter
Allows you to direct messages down a different path of a flow, based on their type.

XSL Transformation

Transforms messages.

- Allows you to perform Extensible Stylesheet Language (XSL) transformations.
- You transform messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the message.

Dynamic routing

You can route messages in various ways using endpoints defined at integration time or endpoints determined, dynamically, at run time.

Dynamic routing covers message routing where the flow is dynamic but all possible endpoints are predefined in a Service Component Architecture (SCA) module, and message routing where the flow is dynamic and the endpoint selection is also dynamic. In the latter case, the service endpoints are selected from an external source, at run time.

Dynamic endpoint selection

The run time has the capability to route request and response messages to an endpoint address identified by a message header element. This message header element can be updated by mediation primitives, in a mediation flow. The endpoint address could be updated with information from a registry, a database, or with information from the message itself. Routing of response messages applies only when the response is being sent by a Web service JAX-WS export.

In order for the run time to implement dynamic routing on a request or response, the SCA module must have the `Use dynamic endpoint if set in the message header` property set. Integration developers can set the `Use dynamic endpoint if set in the message header` property or they can promote it (make it visible at run time), so that the runtime administrator can set it. You can view module properties in the Module Properties window. To see the window, click **Applications** > **SCA Modules** > **Module Properties**. The integration developer gives promoted properties alias names, and these are the names displayed on the administrative console.

Registry

You can use IBM WebSphere Service Registry and Repository (WSRR) to store service endpoint information, and then create SCA modules to retrieve endpoints from the WSRR registry.

When you develop SCA modules, you use the Endpoint Lookup mediation primitive to allow a mediation flow to query a WSRR registry for a service endpoint, or a set of service endpoints. If an SCA module retrieves a set of endpoints then it must use another mediation primitive to select the preferred one.

Mediation policy control of service requests

You can use mediation policies to control mediation flows between service requesters and service providers.

You can control mediation flows using mediation policies stored in IBM WebSphere Service Registry and Repository (WSRR). The implementation of service policy management in WSRR is based on the Web Services Policy Framework (WS-Policy).

In order to control service requests using mediation policies, you must have suitable Service Component Architecture (SCA) modules and mediation policy documents in your WSRR registry.

How to attach a mediation policy to a service request

When you develop an SCA module that needs to make use of a mediation policy, you must include a Policy Resolution mediation primitive in the mediation flow. At run time, the Policy Resolution mediation primitive obtains mediation policy information from the registry. Therefore, an SCA module must contain a mediation flow component in order to support mediation policy control of service requests.

In the registry, you can attach one or more mediation policies to an SCA module, or to a target service used by the SCA module. Attached mediation policies could be used (are in scope) for all service messages processed by that SCA module. The mediation policies can have policy attachments that define conditions. Mediation policy conditions allow different mediation policies to apply in different contexts. In addition, mediation policies can have classifications, which can be used to specify a governance state.

WebSphere Service Registry and Repository

The WebSphere Service Registry and Repository (WSRR) product allows you to store, access, and manage information about service endpoints and mediation policies. You can use WSRR to make your service applications more dynamic, and more adaptable to changing business conditions.

Introduction

Mediation flows can use WSRR as a dynamic lookup mechanism, providing information about service endpoints or mediation policies.

To configure access to WSRR, you create WSRR definition documents using the administrative console. Alternatively, you can use the WSRR administration commands from the wsadmin scripting client. WSRR definitions and their connection properties are the mechanism used to connect to a registry instance, and retrieve a service endpoint or mediation policy.

Service endpoints

You can use WSRR to store information about services that you already use, that you plan to use, or that you want to be aware of. These services might be in your systems, or in other systems. For example, an application could use WSRR to locate the most appropriate service to satisfy its functional and performance needs.

When you develop an SCA module that needs to access service endpoints from WSRR, you must include an Endpoint Lookup mediation primitive in the mediation flow. At run time, the Endpoint Lookup mediation primitive obtains service endpoints from the registry.

Mediation policies

You can also use WSRR to store mediation policy information. Mediation policies can help you to control service requests, by dynamically overriding module properties. If WSRR contains mediation policies that are attached to an object representing either your SCA module or your target service, then the mediation policies could override the module properties. If you want different mediation policies to apply in different contexts, you can create mediation policy conditions.

Note: Mediation policies are concerned with the control of mediation flows, and not with security.

When you develop an SCA module that needs to make use of a mediation policy, you must include a Policy Resolution mediation primitive in the mediation flow. At run time, the Policy Resolution mediation primitive obtains mediation policy information from the registry.

Deployment environments in WebSphere ESB

WebSphere ESB allows you to manage the deployment environment for your Service Component Architecture (SCA) modules as one collection of servers. WebSphere Application Server capabilities included with WebSphere ESB provide elements for this collection of servers.

The WebSphere ESB environment includes a layout of interconnected servers, or topology, which supports SCA modules of your service applications. This topology consists of one server process running on one computer system, or it can consist of multiple server processes running on multiple computer systems. A *server process* is a runtime environment for components that are deployed as SCA modules. In IBM BPM products, including WebSphere ESB, a server process is a Java Virtual Machine (JVM).

If your environment consists of one server process on one system, the server process that is set up is called a *stand-alone server*. A stand-alone server does not have interconnections with other server processes, it has a capacity that is limited to the resources on that one computer system, and it does not include failover support. It is also the easiest environment to set up.

If your environment consists of multiple server processes, you most likely set up those processes as a *clustered* environment in a *cell*. A cell is a management domain of a distributed computing environment consisting of SCA modules and the resources needed to support them. A *deployment environment* is an environment in which server processes, typically on different physical computer systems, are managed together. One deployment manager can manage multiple deployment environments.

Using a deployment environment with clusters provides the following benefits:

- Ease of management: You can have one view for configuring SCA modules, one view of the server processes that support the SCA modules, and one point of control for runtime actions for the SCA modules such as starting, stopping, creating, and deleting.
- Workload balancing: By running application images on multiple servers, a cluster balances an application workload across the servers in the cluster.

- Processing power for the application: You can add processing power to your application by configuring additional server hardware as cluster members supporting the application.
- Application availability: When a server fails, the application continues to process work on the other servers in the cluster thereby allowing recovery efforts to proceed without affecting the application users.
- Maintainability: You can stop a server for planned maintenance without stopping application processing.
- Flexibility: You can add or remove capacity as needed by using the administrative console.

Related tasks:

General steps for implementing a deployment environment
After designing a deployment environment, you will perform specific tasks to make that design a reality. Regardless which method you use to implement the deployment environment, you will perform the same general steps.

Business Space powered by WebSphere

WebSphere ESB includes Business Space powered by WebSphere, which provides a common interface for application users to create, manage and integrate web interfaces across the IBM Business Process Management portfolio.

Business Space is a browser-based graphical user interface that lets application users customize content from products in the IBM Business Process Management portfolio. Application users (business users) are the users of the applications deployed on WebSphere ESB.

Business Space uses mashup technology to enable business users to create and customize their user experience. Mashups are web applications that combine web components (widgets) to create novel interfaces. The widgets communicate with the WebSphere ESB run time using Representational State Transfer (REST) interactions with common web formats, such as JavaScript Object Notation (JSON) and XML data.

Business users can customize Business Space widgets to view the runtime business data according to their preferences. Business Space administrators can create new spaces (collections of pre-configured pages), and mashup page content, in addition to the predefined scenarios that are shipped with Business Space. For example, these mashups act on specific business needs for the enterprise, such as assigning people to various tasks or adjusting business rules for different outcomes.

Business Space is shipped with Process Server, WebSphere ESB, IBM Business Monitor, and WebSphere Business Compass. Templates for predefined scenarios are enabled in Business Space when each product is installed. Business Space also includes information from WebSphere Business Services Fabric that business users can view and modify.

The following diagram shows the Business Space framework and products in the IBM Business Process Management portfolio. The first layer represents Business Space. The second layer contains products that directly contribute content for Business Space: IBM Business Monitor, IBM Process Server, WebSphere Enterprise Service Bus, WebSphere Business Compass, and WebSphere Business Services

Fabric. The third layer represents products that indirectly contribute content through one of the second-layer products: IBM Integration Designer and WebSphere Business Modeler.

Figure 5. Business Space and Business Process Management products

Business Space provided with WebSphere ESB includes templates that you can use to create spaces, or collections of widgets for working with runtime artifacts.

Templates include Solution Administration, Service Administration, and Problem Determination. You create a space using templates by clicking **Manage Spaces** > **Create Space**.

In addition to the widgets related to WebSphere ESB applications, Business Space also includes Google Tools widgets and other widgets for viewing presentations, documents, spreadsheets, web sites, and RSS feeds.

The Business Space framework is installed with WebSphere ESB. For stand-alone server profiles, you can configure Business Space with either the Profile Management Tool or the administrative console. If you use deployment environments or a remote database, you must configure Business Space using the administrative console Business Space Configuration page, or the Deployment Environment Configuration wizard.

After you have installed and configured Business Space, users of your runtime environment can open it from the following URL: http://*host*:*port*/BusinessSpace, where *host* is the name of the host where your server is running and *port* is the port number for your server.

If your team is working with a WebSphere Portal environment, you can configure Business Space for WebSphere Portal. For more information about this configuration, see the related task “Configuring Business Space for WebSphere Portal”.

Related concepts:

Administration widgets

Business Space templates

Related tasks:

Configuring Business Space

Enabling Business Space widget endpoints on the administrative console

Configuring widgets for WebSphere Portal

WebSphere Adapters

WebSphere Adapters provide a service-oriented approach to integration with Enterprise Information Systems (EIS).

WebSphere Adapters are compliant with Java EE Connector Architecture (JCA 1.5). JCA is the Java EE standard for EIS connectivity. EIS Import and EIS Export provide SCA components with the uniform view of the services external to the module. This allows components to communicate with the variety of external EIS systems using the consistent SCA programming model. WebSphere Adapters are assembled in Integration Designer from imported RAR files. They are then exported as an enterprise archive (EAR) file and deployed on WebSphere ESB.

WebSphere Adapters include the following:

- IBM WebSphere Adapter For Email
- IBM WebSphere Adapter For Flat Files
- IBM WebSphere Adapter For FTP
- IBM WebSphere Adapter for JDBC
- IBM WebSphere Adapter for JD Edwards EnterpriseOne
- IBM WebSphere Adapter for Oracle E-Business Suite
- IBM WebSphere Adapter for Siebel Business Applications
- IBM WebSphere Adapter for SAP Software

Figure 6. Detailed schematic of a WebSphere Adapter

Development and deployment of applications on WebSphere ESB

Options for development and deployment of integrated applications on WebSphere ESB include working in the IBM Integration Designer development environment, working with Service Component Architecture APIs, and enabling the applications in a test or production server environment.

Developing service applications

IBM Integration Designer is the separate development environment for WebSphere ESB. In addition to the IBM Integration Designer development environment, Service Component Architecture APIs are published for developers. You can also develop some service components using other application development tools then import them into Integration Designer for modelling, editing, testing, and packaging for deployment into WebSphere ESB.

Within IBM Integration Designer, you can use an assembly editor to group services into *modules* and specify which services are exposed by the module to outside consumers. The modules are then connected to form complete integration solutions. You encapsulate integration logic within modules so that a change to services within a module will not affect any of the other modules in the solution if the interface of the changed module stays the same.

Modules, also called Service Component Architecture (SCA) modules when deployed to WebSphere ESB, determine what artifacts are packaged in enterprise archive (EAR) files that are deployed to the runtime environment.

For more information about developing modules for use with WebSphere ESB, see Developing for WebSphere ESB.

For more information about using IBM Integration Designer to develop integration applications, see the IBM Integration Designer documentation.

Deploying service applications

Deploying is the act of enabling your applications – your SCA modules – in either a test or a production environment. While the concept of deployment is the same for both environments, there are a few differences between the deployment task in each environment. Because it is best to test any changes to your SCA modules on a test server before committing them to the production environment, use IBM Integration Designer to deploy the modules into a test environment, and to package them as a standard enterprise application package, for deployment into WebSphere ESB.

Use WebSphere ESB to install and deploy the applications into a production environment. In WebSphere ESB, you can use the standard WebSphere administrative console to deploy and manage the components of service integration packages. For more information about deploying applications into WebSphere ESB, see Deploying modules.

If you need to deploy many application files, which means installing many SCA modules, you may want to use a batch file. For more information about batch files, see "Deploying applications using Apache Ant tasks".

Related tasks:

"Deploying a mediation module" on page 909
You can deploy a mediation module, as generated by IBM Integration Designer, into a production environment by using these steps.

"Deploying applications using Apache Ant tasks" on page 913
ANT tasks allow you to define the deployment of multiple applications to WebSphere ESB and have them run unattended on a server.

Migration to WebSphere ESB

In this release, you can migrate your installed applications and profile configurations from prior versions of IBM WebSphere ESB to WebSphere ESB version 7.5 using version-to-version migration tools. Version-to-version migration requires that you install the new version of the product alongside the older product, then run migration tools to migrate the existing applications and configurations to the new product. You cannot use updates (in-place upgrades) to migrate from previous releases to WebSphere ESB version 7.5.

Administration of applications on WebSphere ESB

Administering IBM WebSphere ESB involves preparing, monitoring, and modifying the environment into which Service Component Architecture (SCA) modules are deployed as applications and resources, and working with the applications and resources themselves.

For more information about administering applications, see the *Administering Business Process Manager* PDF file.

WebSphere ESB offers several interfaces for administering the runtime environment:

- Administrative console

  The *administrative console* is a browser-based interface where you can monitor, update, stop, and start a wide variety of applications, services, and resources for the applications running on WebSphere ESB. The administrative console can also be used to work with relationships and to locate and resolve failed WebSphere ESB events.

  The administrative console also provide administration capabilities for WebSphere Application Server and other customer-defined products. The WebSphere ESB administrative console is part of the Integrated Solutions Console framework in general, and the WebSphere Application Server administrative console in particular. As a result, many administrative tasks (for example, setting security, viewing logs, and installing applications) are the same for both WebSphere ESB and WebSphere Application Server.

- Command-line tools

  Command-line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks. Using these tools, you can start and stop application servers, check server status, add or remove nodes, and other tasks. The WebSphere ESB command-line tools include the **serviceDeploy** command, which processes .jar, .ear, .war and .rar files exported from a Integration Designer environment and prepares them for installation to the production server.

- WebSphere administrative (wsadmin) scripting program

  The wsadmin scripting program is a non-graphical command interpreter environment that enables you to run administrative options in a scripting language and to submit scripting language programs for execution. It supports the same tasks as the administrative console. The wsadmin tool is intended for production environments and unattended operations.

- Administrative programs

  A set of Java classes and methods under the Java Management Extensions (JMX) specification provide support for administering Service Component Architecture (SCA) and business objects. Each programming interface includes a description of its purpose, an example that demonstrates how to use the interface or class, and references to the individual method descriptions.

Related information:

The administrative console
The administrative console is a browser-based interface used to administer applications, services, and other resources at a cell, node, server, or cluster scope. You can use the console with stand-alone servers and with deployment managers that manage all servers in a cell in a networked environment.

Administering the runtime

Administrative control of mediation processing

You can control mediation flows between service requesters and service providers, administratively.

You can control mediation flows by changing module properties. The module properties set the values of mediation primitive properties.

Module properties

You can change the properties of Service Component Architecture (SCA) modules that contain mediation flows. You can make changes in the following ways:

- When you install an application.
  - From the administrative console
  - Using an administrative command
- When you administer an application.
  - From the administrative console
  - Using an administrative command

The properties that you can change are those that have been promoted from IBM Integration Designer. Any property that you promote is also a dynamic property (which means that it can be overridden at run time using a mediation policy).

The module properties displayed on the administrative console, can do the following:

- Change the values of properties in a mediation flow.
- Provide default values for mediation flows that use mediation policies. (Promoted property values are used when there are no suitable mediation policy values. For further information, see the mediation policy model.)

Promoted properties always have a name, a type, and a value; you can change the value, administratively.

In addition, promoted properties can belong to a group (property groups are introduced in Version 6.2). Property groups can do the following:

- Separate multiple properties that have the same name. An administrator sets property values inside groups. You might have one group for properties on the request flow, and another group for properties on the response flow.
- Set multiple properties (of the same type) under one name. If the integration developer promotes two properties with the same alias name and group, then the administrator can set their values together. You might have logging for the request flow and the response flow, and want to set both at the same time.
- Map to a namespace in a mediation policy.

Security on WebSphere ESB

IBM WebSphere ESB provides a runtime security infrastructure and mechanisms based on IBM WebSphere Application Server security.

Securing the WebSphere ESB environment involves enabling administrative security, enabling application security, creating profiles with security, and restricting access to critical functions to selected users.

System monitoring on WebSphere ESB

You monitor events in WebSphere ESB to assess problem determination, to tune performance, and to measure the effectiveness of your business processes.

WebSphere ESB event monitoring capabilities include performance monitoring and service component monitoring.

Monitoring performance: Performance measurements are available for service component event points, and are processed through the Performance Monitoring Infrastructure (PMI) and the Tivoli® Performance Viewer.

You can monitor specific performance measurements for a given event, such as the number of times the event is invoked or the length of time it takes for that event to complete from start to finish. You can also monitor events, and later view their contents, either by viewing the events in a log file or by querying the events stored on the event database. In both cases, you can temporarily specify an event point or points to monitor in order to spot problems with the application logic or with system performance.

Monitoring service component events: WebSphere ESB monitoring can capture the data in a service component at a certain event point. These events are formatted in a standard called the Common Base Event. You can have the process server publish these events to the logging facilities, or you can use the more versatile monitoring capabilities of a Common Event Infrastructure server database to store and analyze these events.

Some applications that run on the process server include event points that are monitored continually after the application is deployed. You can do this if you are a business analyst and want to observe the effectiveness of the business processes you have modeled and implemented in the applications you deployed on the process server. This allows you to use products, such as IBM Business Monitor, to create customized panels -- or "dashboards" -- to view key business process metrics.

Related tasks:

Introduction: Monitoring
This section tells you about monitoring and improving the overall performance of the service components in your enterprise service bus (ESB). This section also helps you when troubleshooting problems.

Standards compliance

WebSphere ESB is compliant with several government and industry standards, including accessibility standards, information processing standards, software download security standards, and internet protocol standards.

Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

This product uses standard Windows navigation keys.

Accessibility features for WebSphere ESB

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in WebSphere ESB. The accessibility features include the following functions:

- Keyboard-only operation, except in Business Space powered by WebSphere.
- Interfaces that are commonly used by screen readers.

Operating system features that support accessibility are available when you are using WebSphere ESB.

Tip: The WebSphere ESB Information Center is accessibility-enabled for screen reader software, including IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

This product uses standard Web browser navigation keys.

(For information about supported Web browsers, see the WebSphere Enterprise Service Bus System Requirements at http://www.ibm.com/software/integration/wsesb/sysreqs/.)

Interface information

- Installation

  You can install WebSphere ESB either in graphical or silent form. The silent installation program is recommended for users with accessibility needs.

  For instructions, see Installing the product silently.
- Administration

  The administrative console is the primary interface for interacting with the product. This console is displayed within a standard Web browser. By using an accessible Web browser, such as Microsoft Internet Explorer, administrators are able to:
  - Use screen-reader software and a digital speech synthesizer to hear what is displayed on the screen
  - Use voice recognition software, such as IBM ViaVoice®, to enter data and to navigate the user interface
  - Operate features by using the keyboard instead of the mouse

You can configure and administer product features by using standard text editors and scripted or command-line interfaces instead of the graphical interfaces that are provided.

When appropriate, the documentation for specific product features contains additional information about the accessibility of the features.

Vendor software

This product includes certain third-party software not covered under the IBM license agreement. IBM makes no representation about status of these products regarding the Section 508 of the U.S. Federal Rehabilitation. Contact the vendor for information about the Section 508 status of its products. You can request a U.S. Section 508 Voluntary Product Accessibility Template (VPAT) on the IBM Product accessibility information Web page at www.ibm.com/able/product_accessibility.

Related accessibility information

See the IBM Accessibility Center for more information about the commitment that IBM has to accessibility.

Federal Information Processing Standards

Federal Information Processing Standards (FIPS) are standards and guidelines issued by the National Institute of Standards and Technology (NIST) for federal government computer systems.

WebSphere ESB relies on IBM WebSphere Application Server for all cryptographic functions, which are compliant with Federal Information Processing Standards.

FIPS are developed when there are compelling federal government requirements for standards, such as for security and interoperability, but acceptable industry standards or solutions do not exist. Government agencies and financial institutions use these standards to ensure that the products conform to specified security requirements. For more information about these standards, see the National Institute of Standards and Technology at http://www.nist.gov/.

WebSphere Application Server integrates cryptographic modules including Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE), which have undergone FIPS 140-2 certification. In the WebSphere Application Server documentation, the IBM JSSE and JCE modules that have undergone FIPS certification are referred to as IBMJSSEFIPS and IBMJCEFIPS.

For more information, see "Configuring Federal Information Processing Standard Java Secure Socket Extension files"in the WebSphere Application Server information center. When you enable FIPS, several components of the server are affected including the cipher suites, the cryptographic providers, the load balancer, the caching proxy, the high availability manager, and the data replication service.

Related information:

Configuring Federal Information Processing Standard Java Secure Socket Extension files

Internet Protocol Version 6

WebSphere ESB relies on WebSphere Application Server for all Internet Protocol Version 6 compatibility.

IBM WebSphere Application Server Version 7.0 and its JavaMail component support Internet Protocol Version 6 (IPv6).

For more information about this compatibility in WebSphere Application Server, see "JavaMail support for Internet Protocol 6.0" in the WebSphere Application Server Network Deployment documentation.

Related information:

JavaMail support for Internet Protocol 6.0

Globalization

WebSphere ESB is globalized: It has multicultural support, and the user interface and documentation are translated into multiple languages.

Multicultural support means that WebSphere ESB supports the cultural conventions of multiple languages and geographic regions. These conventions include the use of various writing systems and sort orders; various formats for date, time, numbers, and currency; and various keyboard layouts.

Translations are provided for the following national languages:

- Brazilian Portuguese
- Czech
- French
- German
- Hungarian
- Italian
- Japanese
- Korean
- Polish
- Russian
- Simplified Chinese (GB18030 compliant)
- Spanish
- Traditional Chinese

Because WebSphere ESB is built on WebSphere Application Server, you can get information about developing and assembling globalized applications in the WebSphere Application Server information center. See "Learn about WebSphere programming extensions" in the WebSphere Application Server Network Deployment documentation.

Bidirectional language support

WebSphere ESB supports bidirectional languages, through bidirectional enablement. Bidirectional enablement is a mechanism for accurately displaying and processing bidirectional script data inside components either bundled with WebSphere ESB (for example, Web-based tools such as the Common Base Event Browser) or supported by it (for example, service components).

WebSphere ESB processes all bidirectional language data in the logical, left-to-right format, which is the Windows standard bidirectional language format. It processes data passed to internal components, stores data, and outputs the data in that format. WebSphere Adapters and other Enterprise Information Systems (EIS), must convert the data into this format before sending the data to be processed by WebSphere ESB. Because the data output by WebSphere ESB is also in the logical, left-to-right format, the receiving application must convert it to the correct bidirectional format required by the external EIS.

The following table shows the attributes and settings that must match the Windows standard bidirectional format.

Table 2. Bidirectional language format string values

Letter position	Purpose	Allowable values	Default value	Meaning
1	Ordering schema	I	I	Implicit
		V		Visual
2	Orientation	L	L	Left to right
		R		Right to left
		C		Contextual left to right
		D		Contextual right to left

Table 2. Bidirectional language format string values (continued)

Letter position	Purpose	Allowable values	Default value	Meaning
3	Symmetric swapping	Y	Y	Symmetrical swapping is on
		N		Symmetrical swapping is off
4	Shaping	S	N	Text is shaped
		N		Text is not shaped
		I		Initial shaping
		M		Middle shaping
		F		Final shaping
		B		Isolated shaping
5	Numeric	H	N	Hindi (National)
		C		Contextual
		N		Nominal

For data coming from an external component that does not enforce bidirectional support, such as Web services or connectors that are not enabled for processing bidirectional data, you can use example bidirectional application programming interfaces (APIs), based on the IBM Java Development Kit (JDK) to create APIs that transform the data from an external source into the supported bidirectional language format and that transform data sent from WebSphere ESB to an external EIS into the bidirectional format used by that specific EIS.

Note: The locale setting of the user interface (browser) defines the bidirectional language display and edit format.

For more information about bidirectional language, see the technical articles on IBM developerWorks®, available at www.ibm.com/developerworks/websphere/library/techarticles/bidi/bidigen.html.

Related information:

Learn about WebSphere programming extensions

www.ibm.com/developerworks/websphere/library/techarticles/bidi/bidigen.html

Transforming string objects from one bidirectional language format to another

For data coming from an external Enterprise Information System (EIS), you can create APIs that transform string data into the supported bidirectional language format and that transform data sent from WebSphere ESB to an external EIS into the bidirectional format used by that specific EIS.

Before you begin

For more information about bidirectional language support, see Globalization. Use the table in Globalization to determine the correct value for either the input string or output string to use when transforming string data from one format to another.

To create an API to transform the bidirectional language format of string objects, perform the following steps.

Procedure

1. Include all bidirectional classes that contain the bidirectional engine implementation. For example:

   ```
   import com.ibm.bidiTools.bdlayout.*;
   ```

2. Define the strings to contain the data object to transform, and the input and output format values.

   The input format is the bidirectional format in which the string object is currently stored. The output format is the bidirectional format in which you want to store the string object. For example:

   ```
   String strIn = new String("Hello world");
   String formatIn = "ILYNN";
   String formatOut = "VLYNN";
   ```

3. Call the BidiStringTransformation function. For example:

   ```
   String strOut = BiDiStringTransformation(strIn, formatIn, formatOut);
   String BiDiStringTransformation(String strIn, String formatIn, String formatOut) {
   ```

   a. Test if the input string is null. For example:

      ```
      if (strIn == null) return null;
      ```

   b. Perform the transformation. For example:

      ```
      BidiFlagSet flagsIn;
      BidiFlagSet flagsOut;
      formatIn = formatIn.toUpperCase();
      formatOut = formatOut.toUpperCase();

      if (formatIn != null)
          flagsIn = new BidiFlagSet(formatIn.toCharArray());
      else
          flagsIn = new BidiFlagSet();

      if (formatOut != null)
          flagsOut = new BidiFlagSet(formatOut.toCharArray());
      else
          flagsOut = new BidiFlagSet();

      if (flagsIn.equals(flagsOut)) return strIn;
      String strOut = BiDiStringTransformation(strIn, flagsIn, flagsOut);
      return strOut;
    }
      ```

Transforming data objects from one bidirectional language format to another

For data coming from an external Enterprise Information System (EIS), you can create APIs that transform Service Data Objects into the supported bidirectional language format and that transform data sent from WebSphere ESB to an external EIS into the bidirectional format used by that specific EIS.

Before you begin

For more information about bidirectional language support, see Globalization. Use the table in Globalization to determine the correct value for either the input string or output string to use when transforming DataObject-type data from one format to another.

To create an API to transform the bidirectional language format of data objects, perform the following steps.

Procedure

1. Include all bidirectional classes that contain the bidirectional engine implementation. For example:

   ```
   import com.ibm.bidiTools.bdlayout.*;
   ```

2. Include all the classes you need to manipulate the DataObject-type object. For example:

   ```
   import commonj.sdo.DataObject;
   import commonj.sdo.Type;
   import commonj.sdo.Property;
   ```

3. Define string variables to contain the different types of strings that a DataObject-type object contains. This filters the attributes of `type String` while recursively transversing the DataObject. For example:

   ```
   String STRING_STR_TYPE = "String";
   String NORM_STRING_STR_TYPE = "normalizedString";
   String TOKEN_STR_TYPE = "token";
   String LANG_STR_TYPE = "language";
   String NAME_STR_TYPE = "Name";
   String NMTOKEN_STR_TYPE = "NMTOKEN";
   String NCNANE_STR_TYPE = "NCName";
   String ID_STR_TYPE = "ID";
   String IDREF_STR_TYPE = "IDREF";
   String IDREFS_STR_TYPE = "IDREFS";
   String ENTITY_STR_TYPE = "ENTITY";
   String ENTITIES_STR_TYPE = "ENTITIES";
   ```

4. Define the function that verifies if the type of a property is String. For example:

   ```
   private static boolean isStringFamilyType (Property property) {
       boolean rc = false;
       if ((property.getType().getName().equalsIgnoreCase(STRING_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(NORM_STRING_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(TOKEN_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(LANG_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(NAME_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(NMTOKEN_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(NCNANE_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(ID_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(IDREF_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(IDREFS_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(ENTITY_STR_TYPE)) ||
       (property.getType().getName().equalsIgnoreCase(ENTITIES_STR_TYPE)))
          rc = true;
        return rc;
   }
   ```

5. Define the recursive function that applies the bidirectional transformation on the entire DataObject.

   Note: The code logic includes the following assumptions:

   - Bidirectional transformation is applied on the properties of string-type only.
   - The properties of type string in the DataObject are stored in one bidirectional format.

   For example:

   ```
   DataObject BiDiDataObjTransformationBO(DataObject boIn, String formatIn, String formatOut){

      Type type;
      Property property;

      if (boIn == null) return null;

      type = boIn.getType();
   ```

```
List propertyList = type.getProperties();
for (int propertyNumber = 0; propertyNumber < propertyList.size(); propertyNumber++){
      property = (Property) propertyList.get(propertyNumber);
      String propertyName = property.getName();
```

a. Skip all non-string properties. For example:

```
        if (!isStringFamilyType(property))
              continue;

        if (property.isContainment()) {
             if (property.isMany()) {
                   List childsList = boIn.getList(property);
```

b. Recursively call the transformation to handle child objects. For example:

```
                 for (int childNumber = 0; childNumber < childsList.size();
     childNumber++){                    BiDiDataObjTransformationBO(connectionContext,
   ((DataObject)childsList.get(childNumber)),formatIn, formatOut);
                 }
        } else {
```

c. Recursively call the transformation to handle child objects of any contained business objects. For example:

```
                 BiDiDataObjTransformationBO(connectionContext,
       ((DataObject)boIn.get(property)),formatIn, formatOut);
          }
     } else {
```

d. Transform the simple string attributes. For example:

```
          String str = BiDiStringTransformation(
              (boIn.getString(propertyName),formatIn, formatOut);
          boIn.setString(propertyName, str);
      }
   }
   return boIn;
}
```

Chapter 2. Technical overviews of WebSphere ESB

Technical overview information introduces the standards and technical aspects related to the product architecture.

Related information:

PDF documentation
WebSphere Enterprise Service Bus documentation (in PDF format)

Information roadmaps
Business Process Management information roadmaps on IBM developerWorks organize information about WebSphere ESB and the other products in the portfolio.

IBM Education Assistant
Multimedia educational modules about WebSphere ESB, provided by IBM Education Assistant.

Overview
Overview tab, on product library Web page. Use this page to access announcements, data sheets, and other general library documents related to WebSphere ESB.

Service Component Architecture

Service Component Architecture (SCA) enables a service-oriented architecture and is made available by many corporations including IBM. For more details and latest updates, see the Service Component Architecture specification. In this section, we examine SCA services and data objects.

SCA and service invocation

If you take the three aspects of a programming model, which are data, invocation, and composition, and apply some of the new paradigms of a services-based approach, the new programming model for SOA starts to emerge. Service Component Architecture (SCA) provides a way of invoking business services within SOA solutions.

The architectural constructs that make up a service-oriented architecture include a way to represent the data that is exchanged between services, a mechanism for invoking services, and a way to compose services into larger integrated business applications. Today there are many different programming models for supporting each of these. This situation presents developers with the challenge of not only solving a particular business problem, but also choosing and understanding the appropriate implementation technology. One of the important goals of the WebSphere ESB SOA solution is to mitigate these complexities. This is done by converging the various programming models used for implementing service-oriented business applications into a simplified programming model.

This section focuses specifically on the Service Component Architecture (SCA) in WebSphere ESB as the service-oriented component model for defining and invoking business services. SCA plays an important role in providing an invocation model for the SOA solution in WebSphere ESB. SCA also plays a role in composing business services into composite business applications.

© Copyright IBM Corp. 2007, 2011

First, we see that data is primarily represented by Extensible Markup Language (XML) and is programmed with business objects based on the Service Data Object (SDO) specification or through native XML facilities such as XPath or XSLT (Extensible Stylesheet Language Transformation). Second, service invocation maps to Service Component Architecture (SCA). Finally, composition is embodied in process orchestration using Business Process Execution Language (BPEL). The figure shows the three aspects of this new programming model.

Figure 7. Representing data, invocation, and composition within a programming model for SOA

SCA is aimed at providing a simplified programming model for writing applications that run in a Java EE runtime environment, and is based upon concepts and techniques that are refinements of existing Java EE technology. One of the important aspects of SCA is to enable the separation between application business logic and the implementation details. In order to accomplish this, SCA provides a single abstraction for service types that might already be expressed as session beans, Web services, Java classes, or BPEL. The ability to separate business logic from infrastructure logic is important to help reduce the IT resources needed to build an enterprise application, and give developers more time to work on solving a particular business problem rather than focusing on the details of which implementation technology to use.

Modules

A *module* is a unit of deployment that determines which artifacts are packaged together in an Enterprise Archive (EAR) file. Components within a module are collocated for performance, and can pass their data by reference. A module can be seen as a scoping mechanism; that is, it sets an organizational boundary for artifacts.

A module is a composite of service components, imports, and exports. The service components, imports, and exports are stored in the same project and root folder, which also contain the wiring that links the components and the bindings needed for the imports, and exports. A module can also contain the implementations and interfaces referenced by its components, imports and exports, or these may be placed in other projects, such as a library project.

There are two types of modules. A module called module (sometimes referred to as a business integration module) contains a choice of many component types, often used to support a business process. A module called mediation module, contains one or more mediation flow components. It can also contain a component, and one or more Java components that augment the mediation flow component

A module may contain one or more mediation flow components.

The first type of module is primarily designed for business processes. A mediation module is like a gateway to existing external services, which is common in enterprise service bus architectures. These external services or exports are accessed in a mediation module by imports or service providers. By decoupling client service requesters from service providers by a mediation flow, your applications gain flexibility and resilience, a goal of service-oriented architecture. For example, your mediation flow can log incoming messages, route messages to a specific service determined at run time or transform data to make it suitable to pass to another service. These functions can be added and changed over time without modifying the requester or provider services.

A module results in a service application tested and deployed to the IBM Process Server. A mediation module results in a service application tested and deployed to either the IBM Process Server or the WebSphere Enterprise Service Bus server. Both types of modules support imports and exports.

Implementations, interfaces, business objects, business object maps, roles, relationships and other artifacts often need to be shared among modules. A *library* is a project used to store these shared resources.

In the following diagram, the module contains an export, two imports and a service component that uses them. Wiring is shown linking the interfaces and references.

The following list describes module and mediation module artifacts:

Module definition
Defines the module.

Service components
These are definitions of the services in the module. A service component name inside a module is unique. However, a service component can have an arbitrary display name, which is typically a name more useful to a user.

Imports
These are definitions of imports, which are calls to services external to this module. Imports have bindings, which are discussed in the Imports and exports section.

Exports
These are definitions of exports, which are used to expose components to callers that are external to this module. Exports have bindings, which are discussed in the Imports and exports section.

References
These are references from one component to another in the module.

Stand-alone references
These are references to applications that are not defined as Service Component Architecture components (for example, JavaServer Pages), which enable these applications to interact with Service Component Architecture components. There can be only one stand-alone references artifact per module.

Other artifacts
These are artifacts that include WSDL files, Java classes, XSD files, BPEL processes, and so on.

Service components

A service component configures a service implementation. A service component is presented in a standard block diagram.

A component consists of an implementation and one or more interfaces, which defines its inputs, outputs, and faults, and also its references, if applicable. A reference identifies the interface of another service or component that this component requires or consumes. An interface can be defined in a WSDL port type

language or in Java. An interface supports synchronous and asynchronous interaction styles. A component's implementation can be in various languages.

The recommended interface type is WSDL, which is the language used by the provided tutorials and samples. A Java interface, however, is supported and used mostly in the case when a stateless session EJB is imported (discussed later in topics/cimpexp.dita). If you define a component and add the Java implementation later, you should still use a WSDL interface. You cannot mix WSDL-interface-based components with Java interface-based components.

Figure 8. Structure of a component

In Figure 9, there is a component. The implementation, MyValueImpl, and interface of the component are both in Java. The component has two references: another Java interface and a WSDL interface.

Figure 9. Service component structure

When you are working with a component, as shown in the following diagram, you see only the component itself. A reference to this component from another component appears as a line to its interface. A reference from this component would be revealed by a line from its reference point to the interface of other

component. A reference represents a service that this component consumes. When you name a reference and specify its interface only, you allow the component implementation author to defer binding that reference to an actual service until later. The ability to defer binding and reuse implementations is one of the key reasons for using IBM Integration Designer's Service Component Architecture.

A component might also have properties and qualifiers. A qualifier is a quality of service (QoS) directive on interfaces and references for the runtime environment.

Java objects

An implementation of a component in Java is referred to as a Java object. One common implementation is a component that is written in Java. This implementation is sometimes nicknamed a "plain old Java object" or POJO.

Implementations usually use a WSDL interface type; some use a Java interface. If multiple interfaces are specified, then you cannot mix WSDL interfaces with Java interfaces. You can, however, "join" an application created with a set of WSDL interfaces to an application with a set of Java interfaces. A sample listed in the samples gallery of the Welcome view shows you how.

When you work with a Java object, the code remains hidden from you within the context of the editors.

Mediation flow

Mediation is a way of mediating or intervening dynamically between services. A *mediation flow* implements a mediation.

Mediation has several useful functions. For example, you can use mediation when you need to transform data from one service into an acceptable format for a subsequent service. Logging lets you log messages from a service before they are sent to the next service. Routing lets you route data from one service into an appropriate service determined by the mediation flow. A mediation operates independently of the services it connects to. A mediation in the assembly editor appears as a mediation flow component between exports and imports.

In the diagram that follows, three service requesters or exports send their output data to the interface of the mediation flow component. The mediation flow component then routes the appropriate data to two service providers or imports.

Figure 10. Mediation flow component between three service requesters or exports and two service providers or imports

A mediation flow is a flow-like construct created with the mediation flow editor. Selecting a mediation flow component in the assembly editor, launches the mediation flow editor. In the mediation flow editor, an operation from one service, the service requester or export, is mapped to the operation of another service, the service provider or import, along with functions provided by the mediation flow editor. These functions are called *mediation primitives* and are wired in a mediation flow as shown in the following diagram. Mediation primitives are IBM-supplied or you can create your own custom primitives. Mediation primitives can act on both message content and message context, where context is binding-specific information such as SOAP or JMS headers, or user-defined properties.

In the diagram that follows an operation, applyforLoan, sends a message first to a logging primitive, Log, that records the message. Log sends the message to the Filter primitive, which, depending on the message, routes the message to either a processBusinessLoan operation or a processPersonalLoan operation.

Figure 11. Mediation flow between operations

As discussed in the Modules section, mediation flows can be in either a module or a mediation module. Both types of modules can contain one or more mediation

flow components plus zero or more Java components that augment the mediation flow component. A module can be deployed to a WebSphere Process Server. A mediation module can be deployed to either a WebSphere Process Server or a WebSphere Enterprise Service Bus server.

Stand-alone references

Stand-alone references are references to applications that are not defined as Service Component Architecture (SCA) components (for example, JavaServer Pages or servlets). Stand-alone references permit these applications to interact with Service Component Architecture components.

Stand-alone references have neither an interface nor an implementation because the implementation is outside the scope of the module. A module can contain no stand-alone references or one stand-alone references artifact. When you use stand-alone references, you can use your existing applications together with SCA components.

Stand-alone references can be used in a mediation module.

Service data objects

Service data objects complement Service Component Architecture. Service Component Architecture defines the services as components and the connectivity between them. *Service data objects* define the data flowing between components.

Each component passes information as input and output. When a service is invoked, data objects are passed as an XML document with document-literal encoding when using a WSDL port type or as a Java object when using a Java interface. Data objects are the preferred form for data and metadata in Service Component Architecture services. Similar to components, service data objects separate the data object from its implementation. For example, a component interacts with purchase orders while the purchase order itself might use Java Database Connectivity (JDBC) or Enterprise JavaBeans (EJB) to perform the updates to the data. Service data objects let the integration developer focus on working with business artifacts. In fact, service data objects are transparent to the integration developer. For more information about service data objects, see Java Specification Requests JSR 235: Service Data Objects.

In the following diagram, service data objects are passed from an external service to an export, from an export to a component, from a component to a component, from a component to an import, and from an import to a service. Imports and exports are discussed in a subsequent imports and exports section.

Business objects

The computer software industry has developed several programming models and frameworks in which *business objects* provide a natural representation of the business data for application processing.

In general, these business objects:

- Are defined using industry standards
- Transparently map data to database tables or enterprise information systems
- Support remote invocation protocols
- Provide the data programming model foundation for application programming

From a tooling perspective, Integration Designer provides developers with one such common business object model for representing different kinds of business entities from different domains. At development time, this model enables developers to define business objects as XML schema definitions.

At run time, the business data defined by the XML schema definitions is represented as Java Business Objects. In this model, business objects are loosely based on early drafts of the Service Data Object (SDO) specification and provide the complete set of programming model application interfaces required to manipulate business data.

Defining business objects

You define business objects using the business object editor in Integration Designer. The business object editor stores the business objects as XML schema definitions.

Using XML schema to define business objects provides several advantages:

- XML schema provide a standards-based data definition model and a foundation for interoperability between disparate heterogeneous systems and applications. XML schema are used in conjunction with the Web Services Description Language (WSDL) to provide standards-based interface contracts among components, applications, and systems.
- XML schema define a rich data definition model for representing business data. This model includes complex types, simple types, user-defined types, type inheritance, and cardinality, among other features.
- Business objects can be defined by business interfaces and data defined in the Web Services Description Language, as well as by XML schema from industry standards organizations or from other systems and applications. Integration Designer can import these business objects directly.

Integration Designer also provides support for discovering business data in databases and enterprise information systems and then generating the standards-based XML schema business object definition of that business data. Business objects generated in this fashion are often referred to as *application specific business objects* because they mimic the structure of the business data defined in the enterprise information system.

When a process is manipulating data from many different information systems, it can be valuable to transform the disparate representation of business data (for example, CustomerEIS1 and CustomerEIS2 or OrderEIS1 and OrderEIS2) into a single canonical representation (for example, Customer or Order). The canonical representation is often referred to as the *generic business object*.

Business object definitions, particularly for generic business objects, are frequently used by more than one application. To support this reuse, Integration Designer allows business objects to be created in libraries that can then be associated with multiple application modules.

The contracts for the services provided and consumed by a Service Component Architecture (SCA) application module as well as the contracts used to create the components within an application module are defined using the Web Services Description Language. A WSDL can represent both the operations and business objects, which are defined by XML schema to represent the business data, of a contract.

Working with business objects

Service Component Architecture (SCA) provides the framework for defining an application module, the services it provides, the services it consumes, and the composition of components that provide the business logic of the application module. Business objects play an important role in the application, defining the business data that is used to describe the service and component contracts and the business data that the components manipulate.

The following diagram depicts an SCA application module and illustrates many of the places in which the developer works with business objects.

Figure 12. Business objects represent the data that flows between services in an application

Note: This topic describes how business objects are used by SCA application modules. If you are using Java interfaces, the SCA application modules can also process Java objects.

Business object programming model

The business object programming model consists of a set of Java interfaces that represent:

- The business object definition and instance data
- A set of services that support the operations on the business objects

Business object type definitions are represented by the commonj.sdo.Type and commonj.sdo.Property interfaces. The business object programming model provides a set of rules for mapping the XML schema complex type information to the Type interface and each of the elements in the complex type definition to the Property interface.

Business object instances are represented by the commonj.sdo.DataObject interface. The business object programming model is untyped, which means that the same commonj.sdo.DataObject interface can be used to represent different business object definitions, such as Customer and Order. The definition of which properties can be set and retrieved from each business object is determined by type information defined in the XML schema associated with each business object.

The business object programming model behavior is based on the Service Data Object 2.1 specification. For additional information, see the SDO 2.1 for Java specification, tutorials and javadocs on the Web: http://osoa.org/display/Main/Service+Data+Objects+Specifications.

Business object services support various lifecycle operations (such as creation, equality, parsing, and serialization) on business objects.

For specifics on the business object programming model, see Programming using business object services and the Generated API and SPI documentation on business objects.

Bindings, data bindings, and data handlers

As shown in Figure 12 on page 47, business data that is used to invoke services provided by SCA application modules is transformed into business objects so that the SCA components can manipulate the business data. Similarly, the business objects manipulated by SCA components are converted into the data format required by the external services.

In some cases, such as the web service binding, the binding used to export and import services automatically transforms the data into the appropriate format. In other cases, such as the JMS binding, developers can provide a data binding or data handler that converts non-native formats into business objects represented by the DataObject interface.

For more information on developing data bindings and data handlers, refer to "Data handlers" on page 59 and "Data bindings" on page 61.

Components

SCA components define their provision and consumption service contracts using a combination of the Web Services Description Language and XML schema. The business data that SCA passes between components is represented as business objects using the DataObject interface. SCA verifies that these business object types are compatible with the interface contract defined by the component to be invoked.

The programming model abstractions for manipulating business objects vary from component to component. The POJO component and the mediation flow component Custom primitive provide direct manipulating of the business objects by enabling Java programming directly using the business object programming interfaces and services. Most components provide higher level abstractions for manipulating business objects, but also provide snippets of Java code for defining custom behavior in the business object interfaces and services.

Business objects can be transformed using either the combination of the Interface Flow Mediation and Business Object Map component or the mediation flow component and its XML Map primitive. These business object transformation capabilities are useful for converting application specific business objects to and from generic business objects.

Special business objects

Service message objects and business graphs are two specialized types of business objects that are used for specific application purposes.

Service message object

A service message object (SMO) is a specialized business object that is used by mediation flow components to represent the collection of data associated with a service invocation.

A SMO has a fixed top-level structure consisting of headers, context, body, and attachments (if present).

- Headers carry information related to the service invocation over a particular protocol or binding. Examples are SOAP headers and JMS headers.
- Context data carries additional logical information associated with the invocation while it is being processed by the mediation flow component. This information is typically not part of the application data sent or received by clients.
- The body of the SMO carries the payload business data, which represents the core application message or invocation data in the form of a standard business object.

The SMO can also carry attachment data for Web service invocations using SOAP with attachments.

Mediation flows perform such tasks as request routing and data transformation, and the SMO provides the combined view of header and payload contents in a single unified structure.

Business graph

A business graph is a special business object used to provide support for data synchronization in integration scenarios.

Consider an example in which two enterprise information systems have a representation of a specific order. When the order changes in one system, a message can be sent to the other system to synchronize the order data. Business graphs support the notion of sending just the portion of the order that changed to the other system and annotating it with change-summary information to define the type of change.

In this example, an Order business graph would convey to the other system that one of the line items in the order was deleted and that the projected ship date property of the order was updated.

Business graphs can easily be added to existing business objects in Integration Designer. They are most frequently found in scenarios in which WebSphere adapters are being used and to support the migration of WebSphere InterChange Server applications.

Business object parsing mode

Integration Designer provides a property on modules and libraries you can use to configure XML parsing mode for business objects to either eager or lazy.

- If the option is set to *eager*, XML byte streams are eagerly parsed to create the business object.
- If the option is set to *lazy*, the business object is created normally, but the actual parsing of the XML byte stream is deferred and partially parsed only when the business object properties are accessed.

In either XML parsing mode, non-XML data is always eagerly parsed to create the business object.

Considerations when choosing the business object parsing mode

The business object parsing mode determines how XML data is parsed at runtime. A business object parsing mode is defined on a module or library when it is created. You can change the parsing mode of the module or library, however you should be aware of the implications.

The version of the business object runtime framework is set at the module and library level. Modules that were created in a version of IBM Integration Designer prior to version 7 will run in the eager parsing mode without any changes required. By default, modules and libraries that are created in IBM Integration Designer version 7 and later versions will be given the most suitable parsing mode depending on a number of factors such as the parsing mode of existing projects in your workspace, or the parsing mode of dependent projects or other projects in the same solution and so on. You can change the business object parsing mode of a module or library to suit your implementation, however you should be aware of the following considerations.

Considerations

- The lazy business object parsing mode processes XML data faster; however there are compatibility differences between the eager mode and the lazy mode that you need to be aware of before changing the configuration of a module or library. These differences will affect the runtime behavior of the modules. For information on which parsing mode is optimal for your application, see Benefits of using lazy versus eager parsing mode
- A module can only be configured to run in one parsing mode. Libraries can be configured to support either parsing modes or both parsing modes. A library that is configured to support both parsing modes may be referenced by both a module using the eager parsing mode and a module using the lazy parsing mode. The parsing mode of a library an run time is determined by the modules that reference the library. At runtime, a module declares its parsing mode, and that parsing mode is used by the module and any libraries that the module uses.
- Modules and libraries that are configured for different parsing modes are compatible in the following cases:
  - Modules and libraries configured with the lazy parsing mode are compatible with libraries that use either the lazy parsing mode, or both the eager and the lazy parsing modes.
  - Modules and libraries configured with the eager parsing mode are compatible with libraries that use either the eager parsing mode, or both the eager and the lazy parsing modes.
  - Libraries configured with the lazy and eager parsing modes are compatible only with libraries that use both lazy and eager parsing modes.
- Use the same framework for interacting modules that communicate using the SCA binding. If modules communicate using different parsing modes, performance problems may result.

Benefits of using lazy versus eager parsing mode

Some applications benefit from lazy XML parsing mode while others see improved performance with eager parsing mode. It is recommended that you benchmark your application in both parsing modes to determine which mode best suites the specific characteristics of your application.

The following section provides general guidance about the types of applications that benefit from each type of parsing mode:

- Applications benefiting from lazy XML parsing mode

  Applications that parse large XML data streams are likely to see performance improvements when the lazy XML parsing mode is used. The performance benefits increase as the size of the XML byte stream increases and the amount of data from the byte stream that is accessed by the application decreases.
- Applications benefiting from eager parsing mode

  The following applications are likely to perform better in eager parsing mode:
  - Applications that parse non-XML data streams
  - Applications that are created using the BOFactory service
  - Applications that parse very small XML messages

Application migration and development considerations

If you are configuring an application that was originally developed using eager parsing mode to now use lazy parsing mode, or if you are planning to switch an application between lazy and eager parsing mode, be aware of the differences between modes and the considerations when switching modes.

Error handling

If the XML byte stream being parsed is ill-formed, parsing exceptions occur.

- In eager XML parsing mode, those exceptions occur as soon as the business object is parsed from the inbound XML stream.
- If lazy XML parsing mode is configured, the parsing exceptions occur latently when the business object properties are accessed and the portion of the XML that is ill-formed is parsed.

To deal with ill-formed XML, select one of the following options:

- Deploy an enterprise service bus on the edges to validate inbound XML
- Author lazy error-detection logic at the point where business object properties are accessed

Exception stacks and messages

Because the eager and lazy XML parsing modes have different underlying implementations, stack traces thrown by the business object programming interfaces and services have the same exception class name, but they might not contain the same exception message or wrapped set of implementation-specific exception classes.

XML serialization format

The lazy XML parsing mode provides a performance optimization that attempts to copy unmodified XML from the inbound byte stream to the outbound byte stream upon serialization. The result is increased performance, but the serialization format of the outbound XML byte stream might be different if the entire business object was updated in lazy XML parsing mode or if it was running in eager XML parsing mode.

Although the XML serialization format might not be precisely syntactically equivalent, the semantic value provided by the business object is equivalent independent of the parsing modes, and XML can be safely passed between applications running in different parsing modes with semantic equivalence.

Business object instance validator

The lazy XML parsing business object mode instance validator provides a higher fidelity validation of business objects, particularly facet validation of property values. Because of these improvements, the lazy parsing mode instance validator catches additional issues that are not caught in eager parsing mode and provides more detailed error messages.

Version 602 XML Maps

Mediation flows originally developed before WebSphere Integration Developer Version 6.1 might contain XSLT primitives that use a map or stylesheet that cannot execute directly in lazy XML parsing mode. When an application is migrated for use in lazy XML parsing mode, map files associated with XSLT primitives can be automatically updated by the migration wizard to run in the new mode. However, if an XSLT primitive refers directly to a stylesheet that has been edited manually, the stylesheet is not migrated and cannot execute in lazy XML parsing mode.

Private unpublished APIs

If an application is taking advantage of unpublished, private, implementation-specific business object programming interfaces, the application is likely to fail compilation when the parsing mode is switched. In eager parsing mode, these private interfaces are typically business object implementation classes defined by the Eclipse Modeling Framework (EMF).

In all cases, it is strongly recommend that private APIs be removed from the application.

Service Message Object EMF APIs

A mediation component in IBM Integration Designer provides the ability to manipulate message content using the Java classes and interfaces provided in the com.ibm.websphere.sibx.smobo package. In lazy XML parsing mode, the Java interfaces in the com.ibm.websphere.sibx.smobo package can still be used, but methods that refer directly to Eclipse Modeling Framework (EMF) classes and interfaces or that are inherited from EMF interfaces are likely to fail.

The ServiceMessageObject and its contents cannot be cast to EMF objects in lazy XML parsing mode.

BOMode service

The BOMode service is used to determine whether the currently executing XML parsing mode is eager or lazy.

Migration

All applications before version 7.0.0.0 are running in eager XML parsing mode. When they are runtime migrated using the BPM runtime migration tools, they continue to run in eager XML parsing mode.

To enable an application earlier than version 7.0.0.0 to be configured to use the lazy XML parsing mode, you first use Integration Designer to migrate the artifacts of the application. After migration, you then configure the application to use lazy XML parsing.

See Migrating source artifacts for information on migrating artifacts in Integration Designer, and see Configuring the business object parsing mode of modules and libraries for information on setting the parsing mode.

Bindings

At the core of a service-oriented architecture is the concept of a *service*, a unit of functionality accomplished by an interaction between computing devices. An *export* defines the external interface (or access point) of a module, so that Service Component Architecture (SCA) components within the module can provide their services to external clients. An *import* defines an interface to services outside a module, so the services can be called from within the module. You use protocol-specific *bindings* with imports and exports to specify the means of transporting the data into or out of the module.

Exports

External clients can invoke SCA components in an integration module over a variety of protocols (such as HTTP, JMS, MQ, and RMI/IIOP) with data in a variety of formats (such as XML, CSV, COBOL, and JavaBeans). Exports are components that receive these requests from external sources and then invoke WebSphere ESB components using the SCA programming model.

For example, in the following figure, an export receives a request over the HTTP protocol from a client application. The data is transformed into a business object, the format used by the SCA component. The component is then invoked with that data object.

Figure 13. An export with HTTP binding

Imports

An SCA component might want to invoke a non-SCA external service that expects data in a different format. An import is used by the SCA component to invoke the external service using the SCA programming model. The import then invokes the target service in the way that the service expects.

For example, in the following figure, a request from an SCA component is sent, by the import, to an external service. The business object, which is the format used by the SCA component, is transformed to the format expect by the service, and the service is invoked.

Figure 14. An import with HTTP binding

List of bindings

You use IBM Integration Designer to generate a binding for an import or export and to configure the binding. The types of bindings that are available are described in the following list.

- SCA

  The SCA binding, which is the default, lets your service communicate with services in other SCA modules. You use an import with an SCA binding to access a service in another SCA module. You use an export with an SCA binding to offer a service to other SCA modules.

- Web service

  A Web service binding lets you access an external service using interoperable SOAP messages and qualities of service. You can also use Web service bindings to include attachments as part of the SOAP message.

  The Web service binding can use a transport protocol of either SOAP/HTTP (SOAP over HTTP) or SOAP/JMS (SOAP over JMS). Regardless of the transport (HTTP or JMS) used to convey the SOAP messages, Web service bindings always handle request/response interactions synchronously.

- HTTP

  The HTTP binding lets you access an external service using the HTTP protocol, where non-SOAP messages are used, or where direct HTTP access is required. This binding is used when you are working with Web services that are based on the HTTP model (that is, services that use well-known HTTP interface operations such as GET, PUT, DELETE, and so on).

- Enterprise JavaBeans (EJB)

  The EJB bindings let SCA components interact with services provided by Java EE business logic running on a Java EE server.

- EIS

  The EIS (enterprise information system) binding, when used with a JCA resource adapter, lets you access services on an enterprise information system or make your services available to the EIS.

- JMS bindings

  Java Message Service (JMS), generic JMS, and WebSphere MQ JMS (MQ JMS) bindings are used for interactions with messaging systems, where asynchronous communication through message queues is critical for reliability.

  An export with one of the JMS bindings watches a queue for the arrival of a message and asynchronously sends the response, if any, to the reply queue. An import with one of the JMS bindings builds and sends a message to a JMS queue and watches a queue for the arrival of the response, if any.

  – JMS

The JMS binding lets you access the WebSphere-embedded JMS provider.

– Generic JMS

  The generic JMS binding lets you access a non-IBM vendor messaging system.

– MQ JMS

  The MQ JMS binding lets you access the JMS subset of a WebSphere MQ messaging system. You would use this binding when the JMS subset of functions is sufficient for your application.

- MQ

  The WebSphere MQ binding lets you communicate with MQ native applications, bringing them into the service oriented architecture framework and providing access to MQ-specific header information. You would use this binding when you need to use MQ native functions.

Export and import binding overview

An export lets you make services in an integration module available to external clients, and an import makes it possible for your SCA components in an integration module to call external services. The binding associated with the export or import specifies the relationship between protocol messages and business objects. It also specifies the way that operations and faults are selected.

Flow of information through an export

An export receives a request, which is intended for the component to which the export is wired, over a specific transport determined by the associated binding (for example, HTTP).

Figure 15. Flow of a request through the export to a component

When the export receives the request, the following sequence of events occurs:

1. For WebSphere MQ bindings only, the header data binding transforms the protocol header into a header data object.

2. The function selector determines the native method name from the protocol message. The native method name is mapped by the export configuration to the name of an operation on the interface of the export.
3. The request data handler or data binding on the method transforms the request to a request business object.
4. The export invokes the component method with the request business object.
   - The HTTP export binding, the Web service export binding, and the EJB export binding invoke the SCA component synchronously.
   - The JMS, Generic JMS, MQ JMS, and WebSphere MQ export bindings invoke the SCA component asynchronously.

Note that an export can propagate the headers and user properties it receives over the protocol, if context propagation is enabled. Components that are wired to the export can then access these headers and user properties. See the "Propagation" topic in the WebSphere Integration Developer information center for more information.

If this is a two-way operation, the component returns a response.

Figure 16. Flow of a response back through the export

The following sequence of steps occurs:

1. If a normal response message is received by the export binding, the response data handler or data binding on the method transforms the business object to a response.

   If the response is a fault, the fault data handler or data binding on the method transforms the fault to a fault response.

   For HTTP export bindings only, if the response is a runtime exception, the runtime exception data handler, if configured, is called.
2. For WebSphere MQ bindings only, the header data binding transforms the header data objects into protocol headers.

3. The export sends the service response over the transport.

Flow of information through an import

Components send requests to services outside the module using an import. The request is sent, over a specific transport determined by the associated binding.

Figure 17. Flow from a component through the import to a service

The component invokes the import with a request business object.

Note:

- The HTTP import binding, the Web service import binding, and the EJB import binding should be invoked synchronously by the calling component.
- The JMS, Generic JMS, MQ JMS, and WebSphere MQ import binding should be invoked asynchronously.

After the component invokes the import, the following sequence of events occurs:

1. The request data handler or data binding on the method transforms the request business object into a protocol request message.
2. For WebSphere MQ bindings only, the header data binding on the method sets the header business object in the protocol header.
3. The import invokes the service with the service request over the transport.

If this is a two-way operation, the service returns a response, and the following sequence of steps occurs:

Figure 18. Flow of a response back through the import

1. For WebSphere MQ bindings only, the header data binding transforms the protocol header into a header data object.
2. A determination is made about whether the response is a fault.
   - If the response is a fault, the fault selector inspects the fault to determine which WSDL fault it maps to. The fault data handler on the method then transforms the fault to a fault response.
   - If the response is a runtime exception, the runtime exception data handler, if configured, is called.
3. The response data handler or binding on the method transforms the response to a response business object.
4. The import returns the response business object to the component.

Export and import binding configuration

One of the key aspects of export and import bindings is data format transformation, which indicates how data is mapped (deserialized) from a native wire format to a business object or how it is mapped (serialized) from a business object to a native wire format. For bindings associated with exports, you can also specify a function selector to indicate which operation should be performed on the data. For bindings associated with exports or imports, you can indicate how faults that occur during processing should be handled.

In addition, you specify transport-specific information on bindings. For example, for an HTTP binding, you specify the endpoint URL. For the HTTP binding, the transport-specific information is described in the “Generating an HTTP import binding” and “Generating an HTTP export binding” topics. You can also find information about other bindings in the information center.

Data format transformation in imports and exports

When an export or import binding is configured in WebSphere Integration Developer, one of the configuration properties that you specify is the data format used by the binding.

- For export bindings, in which a client application sends requests to and receives responses from an SCA component, you indicate the format of the native data. Depending on the format, the system selects the appropriate data handler or data binding to transform the native data to a business object (which is used by the SCA component) and conversely to transform the business object to native data (which is the response to the client application).
- For import bindings, in which an SCA component sends requests to and receives responses from a service outside the module, you indicate the data format of the native data. Depending on the format, the system selects the appropriate data handler or data binding to transform the business object to native data and vice versa.

WebSphere ESB provides a set of predefined data formats and corresponding data handlers or data bindings that support the formats. You can also create your own custom data handlers and register the data format for those data handlers. For more information, see the "Developing data handlers" topic in the WebSphere Integration Developer information center.

- *Data handlers* are protocol-neutral and transform data from one format to another. In WebSphere ESB, data handlers typically transform native data (such as XML, CSV, and COBOL) to a business object and a business object to native data. Because they are protocol-neutral, you can reuse the same data handler with a variety of export and import bindings. For example, you can use the same XML data handler with an HTTP export or import binding or with a JMS export or import binding.
- *Data bindings* also transform native data to a business object (and vice versa), but they are protocol-specific. For example, an HTTP data binding can be used with an HTTP export or import binding only. Unlike data handlers, an HTTP data binding cannot be reused with an MQ export or import binding.

  Note: Three HTTP data bindings (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML, and HTTPServiceGatewayDataBinding) are deprecated as of WebSphere ESB Version 7.0. Use data handlers whenever possible.

As noted earlier, you can create custom data handlers, if necessary. You can also create custom data bindings; however, it is recommended that you create custom data handlers because they can be used across multiple bindings.

Data handlers:

Data handlers are configured against export and import bindings to transform data from one format to another in a protocol-neutral fashion. Several data handlers are provided as part of the product, but you can also create your own data handler, if necessary. You can associate a data handler with an export or import binding at one of two levels: you can associate it with all operations in the interface of the export or import, or you can associate it with a specific operation for the request or response.

Predefined data handlers

You use IBM Integration Designer to specify the data handler that you want to use.

The data handlers that are predefined for your use are listed in the following table, which also describes how each data handler transforms inbound and outbound data.

Note: Except where noted, these data handlers can be used with JMS, Generic JMS, MQ JMS, WebSphere MQ, and HTTP bindings.
See the "Data handlers" topic in the Integration Designer information center for more detailed information.

Table 3. Predefined data handlers

Data handler	Native data to business object	Business object to native data
ATOM	Parses ATOM feeds into an ATOM feed business object.	Serializes an ATOM feed business object to ATOM feeds.
Delimited	Parses delimited data into a business object.	Serializes a business object to delimited data, including CSV.
Fixed Width	Parses fixed-width data into a business object.	Serializes a business object to fixed-width data.
Handled by WTX	Delegates data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is derived by the data handler.	Delegates data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is derived by the data handler.
Handled by WTX Invoker	Delegates the data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is supplied by the user.	Delegates the data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is supplied by the user.
JAXB	Serializes Java beans to a business object using the mapping rules defined by the Java Architecture for XML Binding (JAXB) specification.	Deserializes a business object to Java beans using the mapping rules defined by the JAXB specification.
JAXWS **Note:** The JAXWS data handler can be used only with the EJB binding.	Used by an EJB binding to transform a response Java object or exception Java object to a response business object using the mapping rules defined by the Java API for XML Web Services (JAX-WS) specification.	Used by an EJB binding to transform a business object to the outgoing Java method parameters using the mapping rules defined by the JAX-WS specification.
JSON	Parses JSON data into a business object.	Serializes a business object to JSON data.
Native body	Parses the native bytes, text, map, stream, or object into one of five base business objects (text, bytes, map, stream, or object).	Transforms the five base business objects into byte, text, map, stream, or object.
SOAP	Parses the SOAP message (and the header) into a business object.	Serializes a business object to a SOAP message.
XML	Parses XML data into a business object.	Serializes a business object to XML data.
UTF8XMLDataHandler	Parses UTF-8 encoded XML data into a business object.	Serializes a business object into UTF-8 encoded XML data when sending a message.

Creating a data handler

Detailed information about creating a data handler can be found in the "Developing data handlers" topic in the Integration Designer information center.

Data bindings:

Data bindings are configured against export and import bindings to transform data from one format to another. Data bindings are specific to a protocol. Several data bindings are provided as part of the product, but you can also create your own data binding, if necessary. You can associate a data binding with an export or import binding at one of two levels–you can associate it with all operations in the interface of the export or import, or you can associate it with a specific operation for the request or response.

You use WebSphere Integration Developer to specify which data binding you want to use or to create your own data binding. A discussion of creating data bindings can be found in the "Overview of JMS, MQ JMS and generic JMS bindings" section of the WebSphere Integration Developer information center.

JMS bindings

The following table lists the data bindings that can be used with:

- JMS bindings
- Generic JMS bindings
- WebSphere MQ JMS bindings

The table also includes a description of the tasks that the data bindings perform.

Table 4. Predefined data bindings for JMS bindings

Data binding	Native data to business object	Business object to native data
Serialized Java object	Transforms the Java serialized object into a business object (which is mapped as the input or output type in the WSDL).	Serializes a business object to the Java serialized object in the JMS object message.
Wrapped bytes	Extracts the bytes from the incoming JMS bytes message and wraps them into the JMSBytesBody business object.	Extracts the bytes from the JMSBytesBody business object and wraps them into the outgoing JMS bytes message
Wrapped map entry	Extracts the name, value, and type information for every entry in the incoming JMS map message and creates a list of MapEntry business objects. It then wraps the list into the JMSMapBody business object	Extracts the name, value, and type information from the MapEntry list in the JMSMapBody business object and creates the corresponding entries in the outgoing JMS map message.
Wrapped object	Extracts the object from the incoming JMS object message and wraps it into the JMSObjectBody business object.	Extracts the object from the JMSObjectBody business object and wraps it into the outgoing JMS object message.
Wrapped text	Extracts the text from the incoming JMS text message and wraps it into the JMSTextBody business object.	Extracts the text from the JMSTextBody business object and wraps it into the outgoing JMS text message.

WebSphere MQ bindings

The following table lists the data bindings that can be used with WebSphere MQ and describes the tasks that the data bindings perform.

Table 5. Predefined data bindings for WebSphere MQ bindings

Data binding	Native data to business object	Business object to native data
Serialized Java object	Transforms the Java serialized object from the incoming message into a business object (which is mapped as the input or output type in the WSDL).	Transforms a business object to the Java serialized object in the outgoing message
Wrapped bytes	Extracts the bytes from the unstructured MQ bytes message and wraps them into the JMSBytesBody business object.	Extracts the bytes from a JMSBytesBody business object and wraps the bytes into the outgoing unstructured MQ bytes message.
Wrapped text	Extracts the text from an unstructured MQ text message and wraps it into a JMSTextBody business object.	Extracts text from a JMSTextBody business object and wraps it in an unstructured MQ text message.
Wrapped stream entry	Extracts the name and type information for every entry in the incoming JMS stream message and creates a list of the StreamEntry business objects. It then wraps the list into the JMSStreamBody business object.	Extracts the name and type information from the StreamEntry list in the JMSStreamBody business object and creates corresponding entries in the outgoing JMSStreamMessage.

In addition to the data bindings listed in Table 5, WebSphere MQ also uses header data bindings. See the WebSphere Integration Developer information center for details.

HTTP bindings

The following table lists the data bindings that can be used with HTTP and describes the tasks that the data bindings perform.

Table 6. Predefined data bindings for HTTP bindings

Data binding	Native data to business object	Business object to native data
Wrapped bytes	Extracts the bytes from the body of the incoming HTTP message and wraps them into the HTTPBytes business object.	Extracts the bytes from the HTTPBytes business object and adds them to the body of the outgoing HTTP message.
Wrapped text	Extracts the text from the body of the incoming HTTP message and wraps it into the HTTPText business object.	Extracts the text from the HTTPText business object and adds it to the body of the outgoing HTTP message.

Function selectors in export bindings

A function selector is used to indicate which operation should be performed on the data for a request message. Function selectors are configured as part of an export binding.

Consider an SCA export that exposes an interface. The interface contains two operations–Create and Update. The export has a JMS binding that reads from a queue.

When a message arrives on the queue, the export is passed the associated data, but which operation from the export's interface should be invoked on the wired component? The operation is determined by the function selector and the export binding configuration.

The function selector returns the native function name (the function name in the client system that sent the message). The native function name is then mapped to the operation or function name on the interface associated with the export. For example, in the following figure, the function selector returns the native function name (CRT) from the incoming message, the native function name is mapped to the Create operation, and the business object is sent to the SCA component with the Create operation.

Figure 19. The function selector

If the interface has only one operation, there is no need to specify a function selector.

Several prepackaged function selectors are available and are listed in the sections that follow.

JMS bindings

The following table lists the function selectors that can be used with:

- JMS bindings
- Generic JMS bindings
- WebSphere MQ JMS bindings

Table 7. Predefined function selectors for JMS bindings

Function selector	Description
JMS function selector for simple JMS data bindings	Uses the JMSType property of the message to select the operation name.
JMS header property function selector	Returns the value of the JMS String Property, TargetFunctionName, from the header.

Table 7. Predefined function selectors for JMS bindings (continued)

Function selector	Description
JMS service gateway function selector	Determines if the request is a one-way or two-way operation by examining the JMSReplyTo property set by the client.

WebSphere MQ bindings

The following table lists the function selectors that can be used with WebSphere MQ bindings.

Table 8. Predefined function selectors for WebSphere MQ bindings

Function selector	Description
MQ handleMessage function selector	Returns handleMessage as a value, which is mapped using the export method bindings to the name of an operation on the interface.
MQ uses JMS default function selector	Reads the native operation from the TargetFunctionName property of the folder of an MQRFH2 header.
MQ uses message body format as native function	Finds the Format field of the last header and returns that field as a String.
MQ type function selector	Creates a method in your export binding by retrieving a URL containing the Msd, Set, Type and Format properties found in the MQRFH2 header.
MQ service gateway function selector	Uses the MsgType property in the MQMD header to determine the operation name.

HTTP bindings

The following table lists the function selectors that can be used with HTTP bindings.

Table 9. Predefined function selectors for HTTP bindings

Function selector	Description
HTTP function selector based on the TargetFunctionName header	Uses the TargetFunctionName HTTP header property from the client to determine which operation to invoke at runtime from the export.
HTTP function selector based on the URL and HTTP method	Uses the relative path from the URL appended with the HTTP method from the client to determine the native operation defined on the export.
HTTP service gateway function selector based on URL with an operation name	Determines the method to invoke based on the URL if "operationMode = oneWay" has been appended to the request URL.

Note: You can also create your own function selector, using IBM Integration Designer. Information about creating a function selector is provided in the IBM Integration Designer information center. For example, a description of creating a function selector for WebSphere MQ bindings can be found in “Overview of the MQ function selectors”.

Fault handling

You can configure your import and export bindings to handle faults (for example, business exceptions) that occur during processing by specifying fault data handlers. You can set up a fault data handler at three levels–you can associate a fault data handler with a fault, with an operation, or for all operations with a binding.

A fault data handler processes fault data and transforms it into the correct format to be sent by the export or import binding.

- For an export binding, the fault data handler transforms the exception business object sent from the component to a response message that can be used by the client application.
- For an import binding, the fault data handler transforms the fault data or response message sent from a service into an exception business object that can be used by the SCA component.

For import bindings, the binding calls the fault selector, which determines whether the response message is a normal response, a business fault, or a runtime exception.

You can specify a fault data handler for a particular fault, for an operation, and for all operations with a binding.

- If the fault data handler is set at all three levels, the data handler associated with a particular fault is called.
- If fault data handlers are set at the operation and binding levels, the data handler associated with the operation is called.

Two editors are used in IBM Integration Designer to specify fault handling. The interface editor is used to indicate whether there will be a fault on an operation. After a binding is generated with this interface, the editor in the properties view lets you configure how the fault will be handled. For more information, see the "Fault selectors" topic in the IBM Integration Designer information center.

How faults are handled in export bindings:

When a fault occurs during the processing of the request from a client application, the export binding can return the fault information to the client. You configure the export binding to specify how the fault should be processed and returned to the client.

You configure the export binding using IBM Integration Designer.

During request processing, a client invokes an export with a request, and the export invokes the SCA component. During the processing of the request, the SCA component can either return a business response or can throw a service business exception or a service runtime exception. When this occurs, the export binding transforms the exception into a fault message and sends it to the client, as shown in the following figure and described in the sections that follow.

Figure 20. How fault information is sent from the component through the export binding to the client

You can create a custom data handler or data binding to handle faults.

Business faults

Business faults are business errors or exceptions that occur during processing.

Consider the following interface, which has a createCustomer operation on it. This operation has two business faults defined: CustomerAlreadyExists and MissingCustomerId.

Figure 21. Interface with two faults

In this example, if a client sends a request to create a customer (to this SCA component) and that customer already exists, the component throws a CustomerAlreadyExists fault to the export. The export needs to propagate this business fault back to the calling client. To do so, it uses the fault data handler that is set up on the export binding.

When a business fault is received by the export binding, the following processing occurs:

1. The binding determines which fault data handler to invoke for handling the fault. If the service business exception contains the fault name, the data handler that is set up on the fault is called. If the service business exception does not contain the name of the fault, the fault name is derived by matching the fault types.
2. The binding calls the fault data handler with the data object from the service business exception.
3. The fault data handler transforms the fault data object to a response message and returns it to the export binding.
4. The export returns the response message to the client.

If the service business exception contains the fault name, the data handler that is set up on the fault is called. If the service business exception does not contain the

name of the fault, the fault name is derived by matching the fault types.

Runtime exceptions

A runtime exception is an exception that occurs in the SCA application during the processing of a request that does not correspond to a business fault. Unlike business faults, runtime exceptions are not defined on the interface.

In certain scenarios, you might want to propagate these runtime exceptions to the client application so that the client application can take the appropriate action.

For example, if a client sends a request (to the SCA component) to create a customer and an authorization error occurs during processing of the request, the component throws a runtime exception. This runtime exception has to be propagated back to the calling client so it can take the appropriate action regarding the authorization. This is achieved by the runtime exception data handler configured on the export binding.

Note: You can configure a runtime exception data handler only on HTTP bindings.

The processing of a runtime exception is similar to the processing of a business fault. If a runtime exception data handler was set up, the following processing occurs:

1. The export binding calls the appropriate data handler with the service runtime exception.
2. The data handler transforms the fault data object to a response message and returns it to the export binding.
3. The export returns the response message to the client.

Fault handling and runtime exception handling are optional. If you do not want to propagate faults or runtime exceptions to the calling client, do not configure the fault data handler or runtime exception data handler.

How faults are handled in import bindings:

A component uses an import to send a request to a service outside the module. When a fault occurs during the processing of the request, the service returns the fault to the import binding. You can configure the import binding to specify how the fault should be processed and returned to the component.

You configure the import binding using IBM Integration Designer. You can specify a fault data handler (or data binding), and you also specify a fault selector.

Fault data handlers

The service that processes the request sends, to the import binding, fault information in the form of an exception or a response message that contains the fault data.

The import binding transforms the service exception or response message into a service business exception or service runtime exception, as shown in the following figure and described in the sections that follow.

Figure 22. How fault information is sent from the service through the import to the component

You can create a custom data handler or data binding to handle faults.

Fault selectors

When you configure an import binding, you can specify a fault selector. The fault selector determines whether the import response is an actual response, a business exception, or a runtime fault. It also determines, from the response body or header, the native fault name, which is mapped by the binding configuration to the name of a fault in the associated interface.

Two types of prepackaged fault selectors are available for use with JMS, MQ JMS, Generic JMS, WebSphere MQ, and HTTP imports:

Table 10. Prepackaged fault selectors

Fault selector type	Description
Header-based	Determines whether a response message is a business fault, a runtime exception, or a normal message based on the headers in the incoming response message.
SOAP	Determines whether the response SOAP message is a normal response, business fault, or runtime exception.

The following shows examples of header-based fault selectors and the SOAP fault selector.

- Header-based fault selector

  If an application wants to indicate that the incoming message is a business fault, there must be two headers in the incoming message for business faults, which is shown as follows:

  ```
  Header name = FaultType, Header value = Business
  Header name = FaultName, Header value = <user defined native fault name>
  ```

  If an application wants to indicate that the incoming response message is a runtime exception, then there must be one header in the incoming message, which is shown as follows:

  ```
  Header name = FaultType, Header value = Runtime
  ```

- SOAP fault selector

  A business fault can be sent as part of the SOAP message with the following custom SOAP header. "CustomerAlreadyExists" is the name of the fault in this case.

  ```
  <ibmSoap:BusinessFaultName
  xmlns:ibmSoap="http://www.ibm.com/soap">CustomerAlreadyExists
  <ibmSoap:BusinessFaultName>
  ```

The fault selector is optional. If you do not specify a fault selector, the import binding cannot determine the type of response. The binding therefore treats it as a business response and calls the response data handler or data binding.

You can create a custom fault selector. The steps for creating a custom fault selector are provided in the "Developing a custom fault selector" topic of the IBM Integration Designer information center.

Business faults

A business fault can occur when there is an error in the processing of a request. For example, if you send a request to create a customer and that customer already exists, the service sends a business exception to the import binding.

When a business exception is received by the binding, the processing steps depend on whether a fault selector has been set up for the binding.

- If no fault selector was set up, the binding calls the response data handler or data binding.
- If a fault selector was set up, the following processing occurs:
  1. The import binding calls the fault selector to determine whether the response is business fault, business response, or runtime fault.
  2. If the response is a business fault, the import binding calls the fault selector to provide the native fault name.
  3. The import binding determines the WSDL fault corresponding to the native fault name returned by the fault selector.
  4. The import binding determines the fault data handler that is configured for this WSDL fault.
  5. The import binding calls this fault data handler with the fault data.
  6. The fault data handler transforms the fault data to a data object and returns it to the import binding.
  7. The import binding constructs a service business exception object with the data object and the fault name.
  8. The import returns the service business exception object to the component.

Runtime exceptions

A runtime exception can occur when there is a problem in communicating with the service. The processing of a runtime exception is similar to the processing of a business exception. If a fault selector was set up, the following processing occurs:

1. The import binding calls the appropriate runtime exception data handler with the exception data.
2. The runtime exception data handler transforms the exception data to a service runtime exception object and returns it to the import binding.
3. The import returns the service runtime exception object to the component.

Interoperability between SCA modules and Open SCA services

The IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture (SCA) provides a simple, yet powerful programming model for constructing applications based on the Open SCA specifications. The SCA modules of WebSphere ESB use import and export bindings to interoperate with Open SCA

services developed in a Rational® Application Developer environment and hosted by the WebSphere Application Server Feature Pack for Service Component Architecture.

An SCA application invokes an Open SCA application by way of an import binding. An SCA application receives a call from an Open SCA application by way of an export binding. A list of supported bindings is shown in "Invoking services over interoperable bindings" on page 71.

Invoking Open SCA services from SCA modules

SCA applications developed with IBM Integration Designer can invoke Open SCA applications developed in a Rational Application Developer environment. This section provides an example of invoking an Open SCA service from an SCA module using an SCA import binding.

Figure 23. Component in SCA module invoking Open SCA service

No special configuration is required to invoke an Open SCA service.

To connect to an Open SCA service by way of an SCA import binding, you provide the component name and service name of the Open SCA service in the import binding.

1. To obtain the name of the target component and service from the Open SCA composite, perform the following steps:
   a. Ensure that the **Properties** tab is open by clicking **Window** > **Show View** > **Properties**.
   b. Open the composite editor by double-clicking the composite diagram that contains the component and service. For example, for a component named **customer**, the composite diagram is **customer.composite_diagram**.
   c. Click the target component.
   d. In the **Name** field of the **Properties** tab, note the name of the target component.
   e. Click the service icon associated with the component.
   f. In the **Name** field of the **Properties** tab, note the name of the service.
2. To configure the WebSphere ESB import to connect it to the Open SCA service, perform the following steps:
   a. In IBM Integration Designer, navigate to the **Properties** tab of the SCA import that you want to connect to the Open SCA service.
   b. In the **Module name** field, enter the component name from step 1d.
   c. In the **Export name** field, enter the service name from step 1f.
   d. Save your work by pressing Ctrl+S.

Invoking SCA modules from Open SCA services

Open SCA applications developed in a Rational Application Developer environment can invoke SCA applications developed with IBM Integration Designer. This section provides an example of invoking an SCA module (by way of an SCA export binding) from an Open SCA service.

Figure 24. Open SCA service invoking component in SCA module

To connect to an SCA component by way of an Open SCA reference binding, you provide the module name and export name.

1. To obtain the name of the target module and export, perform the following steps:
   a. In IBM Integration Designer, open the module in the assembly editor by double-clicking the module.
   b. Click the export.
   c. In the **Name** field of the **Properties** tab, note the name of the export.
2. Configure the Open SCA reference that you want to connect to the WebSphere ESB module and export:
   a. In Rational Application Developer, open the composite editor by double-clicking the composite diagram that contains the component and service.
   b. Click the reference icon of the component reference to display the reference properties in the **Properties** tab.
   c. Click the **Binding** tab on the left side of the page.
   d. Click **Bindings** and then click **Add**.
   e. Select the **SCA** binding.
   f. In the **Uri** field, enter the WebSphere ESB module name, followed by a slash ("/"), followed by the export name (which you determined in step 1c).
   g. Click **OK**.
   h. Save your work by pressing Ctrl+S.

Invoking services over interoperable bindings

The following bindings are supported for interoperability with an Open SCA service.

- SCA binding

  In WebSphere ESB, when an SCA module invokes an Open SCA service by way of an SCA import binding, the following invocation styles are supported:
  - Asynchronous (one-way)
  - Synchronous (request/response)

The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

Note that the SCA binding supports transaction and security context propagation.

- Web service (JAX-WS) binding with either the SOAP1.1/HTTP or SOAP1.2/HTTP protocol

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  In addition, the following qualities of service are supported:

  – Web Services Atomic Transaction
  – Web Services Security

- EJB binding

  A Java interface is used to define the interaction between an SCA module and an Open SCA service when the EJB binding is used.

  Note that the EJB binding supports transaction and security context propagation.

- JMS bindings

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  The following JMS providers are supported:

  – WebSphere Platform Messaging (JMS Binding)
  – WebSphere MQ (MQ JMS Binding)

Note: Business Graphs are not interoperable across any SCA bindings and, therefore, are not supported in interfaces used to interoperate with the WebSphere Application Server Feature Pack for Service Component Architecture.

Binding types

You use protocol-specific *bindings* with imports and exports to specify the means of transporting data into or out of a module.

Selecting appropriate bindings

When you are creating an application, you need to know how to select the binding that is most appropriate to the needs of your application.

The bindings available in IBM Integration Designer provide a range of choices. In this list, you can determine which type of binding might be most suitable for the needs of your application.

Consider a *Service Component Architecture (SCA)* binding when these factors are applicable:

- All services are contained in modules; that is, there are no external services.
- You want to separate function into different SCA modules that interact directly with each other.
- The modules are tightly coupled.

Consider a *Web Service* binding when these factors are applicable:

- You need to access an external service over the Internet or provide a service over the Internet.
- The services are loosely coupled.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

- The protocol of the external service you are accessing or the service you want to provide is SOAP/HTTP or SOAP/JMS.

Consider an *HTTP* binding when these factors are applicable:

- You need to access an external service over the Internet or provide a service over the Internet and you are working with other web services such as GET, PUT, and DELETE.
- The services are loosely coupled.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

Consider an *Enterprise JavaBeans (EJB)* binding when these factors are applicable:

- The binding is for an imported service that is itself an EJB or that needs to be accessed by EJB clients.
- The imported service is loosely coupled.
- Stateful EJB interactions are not required.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

Consider an *Enterprise Information Systems (EIS)* binding when these factors are applicable:

- You need to access a service on an EIS system using a resource adapter.
- Synchronous data transmission is preferred over asynchronous.

Consider a *Java Message Service (JMS)* binding when these factors are applicable:

Important: There are several types of JMS bindings. If you expect to exchange SOAP messages using JMS, consider the Web service binding with the SOAP/JMS protocol. See "Web service bindings" on page 74.

- You need to access a messaging system.
- The services are loosely coupled.
- Asynchronous data transmission is preferred over synchronous.

Consider a *Generic Java Message Service (JMS)* binding when these factors are applicable:

- You need to access a non-IBM vendor messaging system.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider an *Message Queue (MQ)* binding when these factors are applicable:

- You need to access a WebSphere MQ messaging system and need to use the MQ native functions.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider an *MQ JMS* binding when these factors are applicable:

- You need to access a WebSphere MQ messaging system but can do so within a JMS context; that is, the JMS subset of functions is sufficient for your application.
- The services are loosely coupled.

- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

SCA bindings

A Service Component Architecture (SCA) binding lets a service communicate with other services in other modules. An import with an SCA binding lets you access a service in another SCA module. An export with an SCA binding lets you offer a service to other modules.

You use WebSphere Integration Developer to generate and configure SCA bindings on imports and exports in SCA modules.

If modules are running on the same server or are deployed in the same cluster, an SCA binding is the easiest and fastest binding to use.

After the module that contains the SCA binding is deployed to the server, you can use the administrative console to view information about the binding or, in the case of an import binding, to change selected properties of the binding.

Web service bindings

A Web service binding is the means of transmitting messages from a Service Component Architecture (SCA) component to a Web service (and vice versa).

Web service bindings overview:

A Web service import binding allows you to call an external Web service from your Service Component Architecture (SCA) components. A Web service export binding allows you to expose your SCA components to clients as Web services.

With a Web service binding, you access external services using interoperable SOAP messages and qualities of service (QoS).

You use Integration Designer to generate and configure Web service bindings on imports and exports in SCA modules. The following types of Web service bindings are available:

- SOAP1.2/HTTP and SOAP1.1/HTTP

  These bindings are based on Java API for XML Web Services (JAX-WS), a Java programming API for creating Web services.

  – Use SOAP1.2/HTTP if your Web service conforms to the SOAP 1.2 specification.

  – Use SOAP1.1/HTTP if your Web service conforms to the SOAP 1.1 specification.

  Important: When you deploy an application with a Web service (JAX-WS) binding, the target server must not have the **Start components as needed** option selected. See “Checking the server configuration” on page 83 for details.

  When you select one of these bindings, you can send attachments with your SOAP messages.

  The Web service bindings work with standard SOAP messages. Using one of the Web service JAX-WS bindings, however, you can customize the way that SOAP messages are parsed or written. For example, you can handle nonstandard elements in SOAP messages or apply additional processing to the SOAP message. When you configure the binding, you specify a custom data handler that performs this processing on the SOAP message.

You can use policy sets with a Web service (JAX-WS) binding. A policy set is a collection of policy types, each of which provides a quality of service (QoS). For example, the WSAddressing policy set provides a transport-neutral way to uniformly address Web services and messages. You use Integration Designer to select the policy set for the binding.

Note: If you want to use a Security Assertion Markup Language (SAML) policy set, you must perform some additional configuration, as described in "Importing SAML policy sets" on page 81.

- SOAP1.1/HTTP

  Use this binding if you want to create Web services that use a SOAP-encoded message based on Java API for XML-based RPC (JAX-RPC).
- SOAP1.1/JMS

  Use this binding to send or receive SOAP messages using a Java Message Service (JMS) destination.

Regardless of the transport (HTTP or JMS) that is used to convey the SOAP message, Web service bindings always handle request/response interactions synchronously. The thread making the invocation on the service provider is blocked until a response is received from the provider. See "Synchronous invocation" for more information about this invocation style.

Important: The following combinations of Web service bindings cannot be used on exports in the same module. If you need to expose components using more than one of these export bindings, you need to have each in a separate module and then connect those modules to your components using the SCA binding:

- SOAP 1.1/JMS and SOAP 1.1/HTTP using JAX-RPC
- SOAP 1.1/HTTP using JAX-RPC and SOAP 1.1/HTTP using JAX-WS
- SOAP 1.1/HTTP using JAX-RPC and SOAP 1.2/HTTP using JAX-WS

After the SCA module that contains the Web service binding is deployed to the server, you can use the administrative console to view information about the binding or to change selected properties of the binding.

Note: Web services allow applications to interoperate by using standard descriptions of services and standard formats for the messages they exchange. For example, the Web service import and export bindings can interoperate with services that are implemented using Web Services Enhancements (WSE) Version 3.5 and Windows Communication Foundation (WCF) Version 3.5 for Microsoft .NET. When interoperating with such services, you must ensure that:

- The Web Services Description Language (WSDL) file that is used to access a Web service export includes a non-empty SOAP action value for each operation in the interface.
- The Web service client sets either the SOAPAction header or the wsa:Action header when sending messages to a Web service export.

SOAP header propagation:

When handling SOAP messages, you might need to access information from certain SOAP headers in messages that are received, ensure that messages with SOAP headers are sent with specific values, or allow SOAP headers to pass across a module.

When you configure a Web service binding in Integration Designer, you can indicate that you want SOAP headers to be propagated.

- When requests are received at an export or responses received at an import, the SOAP header information can be accessed, allowing logic in the module to be based on header values and allowing those headers to be modified.
- When requests are sent from an export or responses sent from an import, SOAP headers can be included in those messages.

The form and presence of the propagated SOAP headers might be affected by policy sets configured on the import or export, as explained in Table 11 on page 77.

To configure the propagation of SOAP headers for an import or export, you select (from the Properties view of Integration Designer) the **Propagate Protocol Header** tab and select the options you require.

WS-Addressing header

The WS-Addressing header can be propagated by the Web service (JAX-WS) binding.

When you propagate the WS-Addressing header, be aware of the following information:

- If you enable propagation for the WS-Addressing header, the header will be propagated into the module in the following circumstances:
  - When requests are received at an export
  - When responses are received at an import
- The WS-Addressing header is not propagated into outbound messages from WebSphere ESB (that is, the header is not propagated when requests are sent from an import or when responses are sent from the export).

WS-Security header

The WS-Security header can be propagated by both the Web service (JAX-WS) binding and the Web service (JAX-RPC) binding.

The Web services WS-Security specification describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

When you propagate the WS-Security header, be aware of the following information:

- If you enable propagation for the WS-Security header, the header will be propagated across the module in the following circumstances:
  - When requests are received at an export
  - When requests are sent from an import
  - When responses are received at an import
- The header will *not*, by default, be propagated when responses are sent from the export. However, if you set the JVM property `WSSECURITY.ECHO.ENABLED` to `true`, the header will be propagated when responses are sent from the export. In this case, if the WS-Security header on the request path is not modified, WS-Security headers might be automatically echoed from requests into responses.

- The exact form of the SOAP message sent from an import for a request or from an export for a response might not exactly match the SOAP message that was originally received. For this reason, any digital signature should be assumed to become invalid. If a digital signature is required in messages that are sent, it must be established using the appropriate security policy set, and WS-Security headers relating to digital signature in received messages should be removed within the module.

To propagate the WS-Security header, you must include the WS-Security schema with the application module. See "Including the WS-Security schema in an application module" on page 78 for the procedure to include the schema.

How headers are propagated

The way that headers are propagated depends on the security policy setting on the import or export binding, as shown in the Table 11:

Table 11. How security headers are passed

	Export binding with no security policy	**Export binding with security policy**
Import binding with no security policy	Security headers are passed as-is through the module. They are not decrypted. The headers are sent outbound in the same form in which they were received. The digital signature might become invalid.	Security headers are decrypted and passed though the module with signature verification and authentication. The decrypted headers are sent outbound. The digital signature might become invalid.
Import binding with security policy	Security headers are passed as-is through the module. They are not decrypted. The headers should not be propagated to the import. Otherwise, an error occurs because of duplication.	Security headers are decrypted and passed though the module with signature verification and authentication. The headers should not be propagated to the import. Otherwise, an error occurs because of duplication.

Configure appropriate policy sets on the export and import bindings, because this isolates the service requester from changes to the configuration or QoS requirements of the service provider. Having standard SOAP headers visible in a module can then be used to influence the processing (for example, logging and tracing) in the module. Propagating SOAP headers across a module from a received message to a sent message does mean that the isolation benefits of the module are reduced.

Standard headers, such as WS-Security headers, should not be propagated on a request to an import or response to an export when the import or export has an associated policy set that would normally result in the generation of those headers. Otherwise, an error will occur because of a duplication of the headers. Instead, the headers should be explicitly removed or the import or export binding should be configured to prevent propagation of protocol headers.

Accessing SOAP headers

When a message that contains SOAP headers is received from a Web service import or export, the headers are placed in the headers section of the service message object (SMO). You can access the header information, as described in "Accessing SOAP header information in the SMO".

Including the WS-Security schema in an application module

The following procedure outlines the steps for including the schema in the application module:

- If the computer on which Integration Designer is running has access to the Internet, perform the following steps:
  1. In the Business Integration perspective, select **Dependencies** for your project.
  2. Expand **Predefined Resources** and select either **WS-Security 1.0 schema files** or **WS-Security 1.1 schema files** to import the schema into your module.
  3. Clean and rebuild the project.
- If a computer on which Integration Designer is running does not have Internet access, you can download the schema to a second computer that does have Internet access. You can then copy it to the computer on which Integration Designer is running.
  1. From the computer that has Internet access, download the remote schema:
     a. Click **File** > **Import** > **Business Integration** > **WSDL and XSD**.
     b. Select **Remote WSDL** or **XSD file**.
     c. Import the following schemas:

        http://www.w3.org/2003/05/soap-envelope/

        http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd

        http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd
  2. Copy the schemas to the computer that does not have Internet access.
  3. From the computer that has no Internet access, import the schema:
     a. Click **File** > **Import** > **Business Integration** > **WSDL and XSD**.
     b. Select **Local WSDL** or **XSD file**.
  4. Change the schema locations for oasis-wss-wssecurity_secext-1.1.xsd:
     a. Open the schema at *workplace_location/module_name/*`StandardImportFilesGen/oasis-wss-wssecurity-secext-1.1.xsd`.
     b. Change:

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='http://www.w3.org/2003/05/soap-envelope/'/>
        ```

        to:

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='../w3/_2003/_05/soap_envelope.xsd'/>
        ```
     c. Change:

        ```
        <xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
        schemaLocation='http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd'/>
        ```

        to:

        ```
        <xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
        schemaLocation='../w3/tr/_2002/rec_xmlenc_core_20021210/xenc-schema.xsd'/>
        ```
  5. Change the schema location for oasis-200401-wss-wssecurity-secext-1.0.xsd:

a. Open the schema at *workplace_location/module_name/* `StandardImportFilesGen/oasis-200401-wss-wssecurity-secext-1.0.xsd`.
b. Change:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
```

to:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="../w3/tr/_2002/rec_xmldsig_core_20020212/xmldsig-core-schema.xsd"/>
```

6. Clean and rebuild the project.

Transport header propagation:

When handling SOAP messages, you might need to access information from certain transport headers in messages that are received, ensure that messages with transport headers are sent with specific values, or allow transport headers to pass across a module.

When you configure a Web service binding in Integration Designer, you can indicate that you want transport headers to be propagated.

- When requests are received at an export or responses are received at an import, the transport header information can be accessed, allowing logic in the module to be based on header values and allowing those headers to be modified.
- When responses are sent from an export or requests are sent from an import, transport headers can be included in those messages.

Specifying propagation of headers

To configure the propagation of transport headers for an import or export, perform the following steps:

1. From the Properties view of Integration Designer, select **Binding** > **Propagation**.
2. Set the transport header propagation option that you require.

Note: Transport header propagation is disabled by default and can be deployed only to a Version 7.0.0.3 (or later) runtime environment. Also note that, for Version 7.0.0.3, transport header propagation is limited to HTTP transport headers only.

If you enable propagation of transport headers, the headers will be propagated across a module from received messages and, if you do not explicitly remove the headers, the headers will be used in subsequent invocations in the same thread.

Note: Transport headers cannot be propagated when you are using the Web service (JAX-RPC) binding.

Accessing the header information

When transport header propagation is enabled for received messages, all transport headers (including customer-defined headers) are visible in the service message object (SMO). You can set the headers to different values or create new ones. Note, however, that there is no checking or validation of the values you set, and any improper or incorrect headers might cause Web service runtime problems.

Consider the following information about the setting of HTTP headers:

- Any changes to the headers that are reserved for the Web service engine will not be honored in the outbound message. For example, the HTTP version or method, Content-Type, Content-Length and SOAPAction headers are reserved for the Web service engine.
- If the header value is a number, the number (rather than the string) should be set directly. For example, use `Max-Forwards = 5` (rather than `Max-Forwards = Max-Forwards: 5`) and `Age = 300` (rather than `Age = Age: 300`).
- If the request message is less than 32 KB in size, the Web service engine removes the Transfer-Encoding header and instead sets the Content-Length header to the fixed size of the message.
- The Content-language is reset by WAS.channel.http on the response path.
- An invalid setting for Upgrade results in a 500 error.
- The following headers append the value reserved by the Web service engine to the customer settings:
  - User-Agent
  - Cache-Control
  - Pragma
  - Accept
  - Connection

You can access the header information in one of the following ways:

- Using a mediation primitive to access the SMO structures

  See the "Related Information" links to find information about using mediation primitives.
- Using the context service SPI

  The following sample code reads the HTTP transport headers from the context service:

```
HeadersType headerType = ContextService.INSTANCE.getHeaders();
HTTPHeaderType httpHeaderType = headerType.getHTTPHeader();
List HTTPHeader  httpHeaders = httpHeaderType.getHeader();
if(httpHeaders!=null){
 for(HTTPHeader httpHeader: httpHeaders){
   String httpHeadername = httpHeader.getName();
   String httpHeaderValue = httpHeader.getValue();
     }
}
List PropertyType  properties = headerType.getProperties();
if(properties!=null){
  for(PropertyType property: properties){
   String propertyName = property.getName();
   String propertyValue = property.getValue().toString();
     }
}
```

Troubleshooting

If you encounter problems when sending the revised headers, you can intercept the TCP/IP message by using tools such as the TCP/IP Monitor in Integration Designer. You access the TCP/IP Monitor by selecting **Run/Debug** > **TCP/IP Monitor** from the Preferences page.

You can also view the header values using the JAX-WS engine trace:
`org.apache.axis2.*=all: com.ibm.ws.websvcs.*=all:`

Working with Web service (JAX-WS) bindings:

When you use Web service (JAX-WS) bindings with your applications, you can add a Security Assertion Markup Language (SAML) quality of service (QOS) to the binding. You must first use the administrative console to import the policy set. You can also use the administrative console to make sure that the server is properly configured for use with the Web service (JAX-WS) binding.

Importing SAML policy sets:

Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information. When you configure a Web service (JAX-WS) binding in Integration Designer, you can specify an SAML policy set. You first use the administrative console of WebSphere ESB to make the SAML policy sets available so that they can be imported into Integration Designer.

Before you begin

The SAML policy sets are typically located in the profile configuration directory:

profile_root/config/templates/PolicySets

Before you begin this procedure, verify that the following directories (which contain the policy sets) are located in the profile configuration directory:

- SAML11 Bearer WSHTTPS default
- SAML20 Bearer WSHTTPS default
- SAML11 Bearer WSSecurity default
- SAML20 Bearer WSSecurity default
- SAML11 HoK Public WSSecurity default
- SAML20 HoK Public WSSecurity default
- SAML11 HoK Symmetric WSSecurity default
- SAML20 HoK Symmetric WSSecurity default
- Username WSHTTPS default

If the directories are not in the profile configuration directory, copy them to that directory from the following location:

app_server_root/profileTemplates/default/documents/config/templates/PolicySets

About this task

You import the policy sets into the administrative console, select the ones you want to make available to Integration Designer, and then save a zip file for each of those policy sets to a location that is accessible by Integration Designer.

Procedure

1. Import the policy sets by following these steps:
   a. From the administrative console, click **Services** > **Policy Sets** > **Application policy sets**.
   b. Click **Import** > **From Default Repository**.
   c. Select the SAML default policy sets, and click **OK**.
2. Export the policy sets so that they can be used by Integration Designer:

a. From the Application policy sets page, select the SAML policy set you want to export, and click **Export**.

   Note: If the Application policy sets page is not currently displayed, click **Services** > **Policy Sets** > **Application Policy Sets** from the administrative console.
b. On the next page, click the .zip file link for the policy set.
c. In the File Download window, click **Save** and indicate a location that is accessible by Integration Designer.
d. Click **Back**.
e. Complete steps 2a through 2d for each policy set you want to export.

Results

The SAML policy sets are saved in .zip files and are ready to be imported into Integration Designer.

What to do next

Import the policy sets into Integration Designer, as described in the topic "Policy sets".

Invoking web services that require HTTP basic authentication:

HTTP basic authentication employs a user name and password to authenticate a service client to a secure endpoint. You can set up HTTP basic authentication when sending or receiving web service requests.

About this task

You set up HTTP basic authentication for receiving web service requests by configuring the Java API for XML Web Services (JAX-WS) export binding, as described in the "Creating and assigning security roles to web service exports" topic.

HTTP basic authentication can be enabled for web service requests that are sent by a JAX-WS import binding in one of two ways:

- When configuring the import binding in an SCA module, you can select the supplied HTTP authentication policy set named BPMHTTPBasicAuthentication (which is provided with the web service (JAX-WS) import binding) or any other policy set that includes the HTTPTransport policy.
- When constructing the SCA module, you can use mediation flow capabilities to dynamically create a new HTTP authentication header and specify the user name and password information in the header.

Note: The policy set has precedence over the value specified in the header. If you want to use the value set in the HTTP authentication header at run time, do not attach a policy set that includes the HTTPTransport policy. Specifically, do not use the default BPMHTTPBasicAuthentication policy set, and, if you have defined a policy set, make sure it excludes the HTTPTransport policy.

You can find more information about web service policy sets and policy bindings and how they are used in the "Web services policy sets" topic of the WebSphere Application Server Information Center. The link to the topic is listed under "Related information".

Procedure

- To use the supplied policy set, perform the following steps:
  1. Optional: In the administrative console, create a client general policy binding or edit an existing one that includes the HTTPTransport policy with the desired user ID and password values.
  2. In IBM Integration Designer, generate a web service (JAX-WS) import binding and attach the BPMHTTPBasicAuthentication policy set.
  3. Perform *one* of the following steps:
     - In IBM Integration Designer, in the web service (JAX-WS) import binding properties, specify the name of an existing client general policy binding that includes the HTTPTransport policy.
     - After deploying the SCA module, from the administrative console of the process server, select an existing client policy binding or create a new one and associate it with the import binding.
  4. Optional: In the administrative console of the process server, edit the selected policy set binding to specify the required ID and password.
- To specify the user name and password in the HTTP authentication header, perform one of the following sets of steps:
  - Use the HTTP Header Setter mediation primitive in IBM Integration Designer to create the HTTP authentication header, and specify the user name and password.
  - If additional logic is required, use Java code in a custom mediation primitive (as shown in the following example) to:
    1. Create an HTTP authentication header.
    2. Specify the user name and password information.
    3. Add the new HTTP authentication header to HTTPControl.
    4. Set the updated HTTPControl back in the Context service.

```
//Get the HeaderInfoType from contextService
ContextService contextService = (ContextService) ServiceManager.INSTANCE
.locateService("com/ibm/bpm/context/ContextService");
HeaderInfoType headers = contextService.getHeaderInfo();
if(headers == null){
    headers = ContextObjectFactory.eINSTANCE.createHeaderInfoType();
}
//Get the HTTP header and HTTP Control from HeaderInfoType
HTTPHeaderType httpHeaderType = headers.getHTTPHeader();
HTTPControl cp = httpHeaderType.getControl();
HeadersFactory factory = HeadersFactory.eINSTANCE;
if(cp == null){
    cp = factory.createHTTPControl();
}
//Create new HTTPAuthentication and set the HTTPCredentials
HTTPAuthentication authorization = factory.createHTTPAuthentication();
HTTPCredentials credentials = factory.createHTTPCredentials();
authorization.setAuthenticationType(HTTPAuthenticationType.BASIC_LITERAL);
credentials.setUserId("USERNAME");
credentials.setPassword("PASSWORD");
authorization.setCredentials(credentials);
cp.setAuthentication(authorization);
httpHeaderType.setControl(cp);
// Set header info back to the current execution context.
contextService.setHeaderInfo(headers);
```

Checking the server configuration:

When you deploy an application with a Web service (JAX-WS) binding, you must make sure that the server on which the application is deployed does not have the **Start components as needed** option selected.

About this task

You can check to see whether this option is selected by performing the following steps from the administrative console:

Procedure

1. Click **Servers** > **Server types** > **WebSphere application servers**.
2. Click the server name.
3. From the Configuration tab, determine whether **Start components as needed** is selected.
4. Perform one of the following steps:
   - If **Start components as needed** is selected, remove the check, and then click **Apply**.
   - If **Start components as needed** is not selected, click **Cancel**.

Attachments in SOAP messages:

You can send and receive SOAP messages that include binary data (such as PDF files or JPEG images) as attachments. Attachments can be *referenced* (that is, represented explicitly as message parts in the service interface) or *unreferenced* (in which arbitrary numbers and types of attachments can be included).

A referenced attachment can be represented in one of the following ways:

- MTOM attachments use the SOAP Message Transmission Optimization Mechanism (http://www.w3.org/TR/soap12-mtom/) specified encoding. MTOM attachments are enabled through a configuration option in the import and export bindings and are the recommended way to encode attachments for new applications.
- As a wsi:swaRef-typed element in the message schema

  Attachments defined using the wsi:swaRef type conform to the Web Services Interoperability Organization (WS-I) *Attachments Profile Version 1.0* (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), which defines how message elements are related to MIME parts.
- As a top-level message part, using a binary schema type

  Attachments represented as top-level message parts conform to the *SOAP Messages with Attachments* (http://www.w3.org/TR/SOAP-attachments) specification.

  Attachments represented as top-level message parts can also be configured to ensure that the WSDL document and messages produced by the binding conform to the WS-I *Attachments Profile Version 1.0* and the WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html).

An unreferenced attachment is carried in a SOAP message without any representation in the message schema.

In all cases, except MTOM attachments, the WSDL SOAP binding should include a MIME binding for attachments to be used, and the maximum size of the attachments should not exceed 20 MB.

Note: To send or receive SOAP messages with attachments, you must use one of the Web service bindings based on the Java API for XML Web Services (JAX-WS).

How to choose the appropriate attachment style:

When designing a new service interface that includes binary data, consider how that binary data is carried in the SOAP messages that are sent and received by the service.

Message Transmission Optimization Mechanism (MTOM) should be used for attachments if the connected web service application supports it. If not, the following diagram shows how other attachment styles are chosen. Use the following set of questions to determine the appropriate attachment style:

MTOM attachments: top-level message parts:

You can send and receive web service messages which include SOAP Message Transmission Optimization Mechanism (MTOM) attachments. In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachment or attachments are in subsequent parts.

By sending or receiving a referenced attachment in a SOAP message, the binary data that makes up the attachment (which is often quite large) is held separately

from the SOAP message body so that it does not need to be parsed as XML. This results in more efficient processing than if the binary data were held within an XML element.

The following is a sample of an MTOM SOAP message:

```
... other transport headers ...
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812; type="application/xop+xml"; start="
<0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>"; start-info="text/xml"; charset=UTF-8

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
   <0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>

<?xml version="1.0" encoding="UTF-8"?>
         <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
            <soapenv:Header/>
            <soapenv:Body>
               <sendImage xmlns="http://org/apache/axis2/jaxws/sample/mtom">
                  <input>
                     <imageData><xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/></imageData>
                  </input>
               </sendImage>
            </soapenv:Body>
         </soapenv:Envelope>
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: text/plain
content-transfer-encoding: binary
content-id:
         <1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org>

... binary data goes here ...
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812--
```

Note that in the MTOM sample, the content-type for the SOAP envelope is `application/xop+xml` and the binary data is replaced by an `xop:Include` element like below:

```
<xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/>
```

Inbound processing of referenced attachments

When a client passes a SOAP message with an attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the attachment binary in the business object.

Figure 25. How the Web service (JAX-WS) export binding processes a SOAP message with a referenced attachment

MTOM attachment attributes

- MTOM can support attachment elements within nested structures.
- MTOM is only available for the base64Binary type.
- MTOM can support attachment elements within nested structures meaning that the `bodyPath` for MTOM attachments are the `xpath` location for the element where the MTOM attachment is held. The computing logic for `bodyPath` is strictly following the schema to generate the `xpath` location as shown in the examples below:
  - For a non-array type (`maxOccurs` is 1): /sendImage/input/imageData

– For a array type (`maxOccurs` > 1): /sendImage/input/imageData[1]

- Mixed attachment types are not supported, meaning that if MTOM is enabled on the import binding, the MTOM attachment will be generated. If MTOM is disabled or if the MTOM configuration value is left as the default value on the export binding, the incoming MTOM message is not supported.

Referenced attachments: swaRef-typed elements:

You can send and receive SOAP messages that include attachments represented in the service interface as swaRef-typed elements.

An swaRef-typed element is defined in the Web Services Interoperability Organization (WS-I) *Attachments Profile* Version 1.0 (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), which defines how message elements are related to MIME parts.

In the SOAP message, the SOAP body contains an swaRef-typed element that identifies the content ID of the attachment.

Figure 26. A SOAP message with an swaRef element

The WSDL for this SOAP message contains an swaRef-typed element within a message part that identifies the attachment.

```
<element name="sendPhoto">
  <complexType>
     <sequence>
       <element name="Photo" type="wsi:swaRef"/>
     </sequence>
  </complexType>
 </element>
```

The WSDL should also contain a MIME binding that indicates MIME multipart messages are to be used.

Note: The WSDL does *not* include a MIME binding for the specific swaRef-typed message element, because MIME bindings apply only to top-level message parts.

Attachments represented as swaRef-typed elements can be propagated only across mediation flow components. If an attachment must be accessed by or propagated to another component type, use a mediation flow component to move the attachment to a location that is accessible by that component.

Inbound processing of attachments

You use Integration Designer to configure an export binding to receive the attachment. You create a module and its associated interface and operations, including an element of type swaRef. You then create a Web service (JAX-WS) binding.

Note: See the "Working with attachments" topic in the Integration Designer information center for more detailed information.

When a client passes a SOAP message with an swaRef attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the content ID of the attachment in the business object.

Figure 27. How the Web service (JAX-WS) export binding processes a SOAP message with an swaRef attachment

Accessing attachment metadata in a mediation flow component

As shown in Figure 28 on page 89, when swaRef attachments are accessed by components, the attachment content identifier appears as an element of type swaRef.

Each attachment of a SOAP message also has a corresponding `attachments` element in the SMO. When using the WS-I swaRef type, the `attachments` element includes the attachment content type and content ID as well as the actual binary data of the attachment.

To obtain the value of an swaRef attachment, it is therefore necessary to obtain the value of the swaRef-typed element, and then locate the `attachments` element with the corresponding `contentID` value. Note that the `contentID` value typically has the `cid:` prefix removed from the swaRef value.

Figure 28. How swaRef attachments appear in the SMO

Outbound processing

You use Integration Designer to configure a Web service (JAX-WS) import binding to invoke an external Web service. The import binding is configured with a WSDL document that describes the Web service to be invoked and defines the attachment that will be passed to the Web service.

When an SCA message is received by a Web service (JAX-WS) import binding, swaRef-typed elements are sent as attachments if the import is wired to a mediation flow component and the swaRef-typed element has a corresponding `attachments` element.

For outbound processing, swaRef-typed elements are always sent with their content ID values; however the mediation module must ensure that there is a corresponding `attachments` element with a matching `contentID` value.

Note: To conform to the WS-I Attachments Profile, the `content ID` value should follow the "content-id part encoding," as described in section 3.8 of the WS-I *Attachments Profile* 1.0.

Figure 29. How the Web service (JAX-WS) import binding generates a SOAP message with an swaRef attachment

Setting attachment metadata in a mediation flow component

If, in the SMO, there is an swaRef-typed element value and an `attachments` element, the binding prepares the SOAP message (with the attachment) and sends it to a recipient.

Figure 30. How an swaRef attachment in the SMO is accessed to create the SOAP message

The `attachments` element is present in the SMO only if a mediation flow component is connected directly to the import or export; it does not get passed across other component types. If the values are needed in a module that contains other component types, a mediation flow component should be used to copy the

values into a location where they can then be accessed in the module, and another mediation flow component used to set the correct values before an outbound invocation by way of a Web service import.

Important: As described in "XML representation of SMO," the XSL Transformation mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The XSL Transformation mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Referenced attachments: top-level message parts:

You can send and receive SOAP messages that include binary attachments that are declared as parts in your service interface.

In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachment or attachments are in subsequent parts.

What is the advantage of sending or receiving a referenced attachment in a SOAP message? The binary data that makes up the attachment (which is often quite large) is held separately from the SOAP message body so that it does not need to be parsed as XML. This results in more efficient processing than if the binary data were held within an XML element.

Types of SOAP messages with referenced attachments

Beginning with Version 7.0.0.3 of WebSphere ESB, you have a choice about how the SOAP message is generated:

- **WS-I-compliant messages**

  The runtime can generate SOAP messages that comply with the WS-I *Attachments Profile Version 1.0* and the WS-I *Basic Profile Version 1.1*. In a SOAP message that is compliant with these profiles, only one message part is bound to the SOAP body; for those that are bound as attachments, the content-id part encoding (as described in the WS-I *Attachments Profile Version 1.0*) is used to relate the attachment to the message part.
- **Non-WS-I-compliant messages**

  The runtime can generate SOAP messages that do not comply with the WS-I profiles but that are compatible with the messages generated in Version 7.0 or 7.0.0.2 of WebSphere ESB. The SOAP messages use top-level elements named after the message part with an `href` attribute that holds the attachment `content-id`, but the content-id part encoding (as described in the WS-I *Attachments Profile Version 1.0*) is not used.

Selecting WS-I compliance for Web service exports

You use Integration Designer to configure an export binding. You create a module and its associated interface and operations. You then create a Web service (JAX-WS) binding. The Referenced attachments page displays all the binary parts

from the created operation, and you select which parts will be attachments. You then specify, on the Specify the WS-I AP 1.0 compliance page of Integration Designer, one of the following choices:

- **Use WS-I AP 1.0 compliant SOAP message**

  If you select this option, you also specify which message part should be bound to the SOAP body.

  Note: This option can be used only when the corresponding WSDL file is also WS-I compliant.

  A WSDL file that is generated by Integration Designer Version 7.0.0.3 is compliant with WS-I. However, if you import a WSDL file that is not compliant with WS-I, you cannot select this option.
- **Use non WS-I AP 1.0 compliant SOAP message**

  If you select this option, which is the default, the first message part is bound to the SOAP body.

Note: Only top-level message parts (that is, elements defined in the WSDL portType as parts within the input or output message) that have a binary type (either base64Binary or hexBinary) can be sent or received as referenced attachments.
See the "Working with attachments" topic in the Integration Designer information center for more detailed information.

For WS-I-compliant messages, the content-ID that is generated in the SOAP message is a concatenation of the following elements:

- The value of the `name` attribute of the `wsdl:part` element referenced by the `mime:content`
- The character `=`
- A globally unique value, such as a UUID
- The character `@`
- A valid domain name

Inbound processing of referenced attachments

When a client passes a SOAP message with an attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the attachment binary in the business object.

Figure 31. How the Web service (JAX-WS) export binding processes a WS-I compliant SOAP message with a referenced attachment

Accessing attachment metadata in a mediation flow component

As shown in Figure 31, when referenced attachments are accessed by components, the attachment data appears as a byte array.

Each referenced attachment of a SOAP message also has a corresponding `attachments` element in the SMO. The `attachments` element includes the attachment content type and the path to the message body element where the attachment is held.

Figure 32. How referenced attachments appear in the SMO

Important: The path to the message body element is not automatically updated if the message is transformed and the attachment moved. You can use a mediation flow to update the `attachments` element with the new path (for example, as part of the transform or using a separate message element setter).

How outbound SOAP messages are constructed

You use Integration Designer to configure a Web service (JAX-WS) import binding to invoke an external Web service. The import binding is configured with a WSDL document that describes the Web service to be invoked and defines which message parts should be passed as attachments. You can also indicate, on the Specify the WS-I AP 1.0 compliance page of Integration Designer, one of the following choices:

- **Use WS-I AP 1.0 compliant SOAP message**

  If you select this option, you also specify which message part should be bound to the SOAP body; all others are bound to attachments or headers. Messages sent by the binding do not include elements in the SOAP body that refer to the attachments; the relationship is expressed by way of the attachment content ID including the message part name.
- **Use non WS-I AP 1.0 compliant SOAP message**

  If you select this option, which is the default, the first message part is bound to the SOAP body; all others are bound to attachments or headers. Messages sent by the binding include one or more elements in the SOAP body that refer to the attachments by way of an `href` attribute.

Note: The part that represents an attachment, as defined in the WSDL, must be a simple type (either base64Binary or hexBinary). If a part is defined by a complexType, that part cannot be bound as an attachment.

Outbound processing of referenced attachments

The import binding uses information in the SMO to determine how the binary top-level message parts are sent as attachments.

Figure 33. How the referenced attachment in the SMO is accessed to create the SOAP message

The `attachments` element is present in the SMO only if a mediation flow component is connected directly to the import or export; it does not get passed across other component types. If the values are needed in a module that contains other component types, a mediation flow component should be used to copy the values into a location where they can then be accessed in the module, and another mediation flow component used to set the correct values before an outbound invocation by way of a Web service import.

The binding uses a combination of the following conditions to determine how (or whether) the message is sent:

- Whether there is a WSDL MIME binding for the top-level binary message part and, if so, how the content type is defined
- Whether there is an `attachments` element in the SMO whose `bodyPath` value references a top-level binary part

How attachments are created when an `attachment` element exists in the SMO

The following table shows how an attachment is created and sent if the SMO contains an `attachment` element with a `bodyPath` that matches a message name part:

Table 12. How the attachment is generated

Status of WSDL MIME binding for top-level binary message part	How message is created and sent
Present with one of the following: • No defined content type for the message part • Multiple content types defined • Wildcard content type defined	Message part is sent as an attachment. `Content-Id` is set to the value in the attachments element if present; otherwise, one is generated. `Content-Type` is set to the value in the attachments element if present; otherwise, it is set to `application/octet-stream`.
Present with single, non-wildcard content for the message part	Message part is sent as an attachment. `Content-Id` is set to the value in the attachments element if present; otherwise, one is generated. `Content-Type` is set to the value in the attachments element if present; otherwise, it is set to the type defined in the WSDL MIME content element.
Not present	Message part is sent as an attachment. `Content-Id` is set to the value in the attachments element if present; otherwise, one is generated. `Content-Type` is set to the value in the attachments element if present; otherwise, it is set to `application/octet-stream`. **Note:** Sending message parts as attachments when not defined as such in the WSDL may break compliance with the WS-I Attachments Profile 1.0 and so should be avoided if possible.

How attachments are created when no `attachment` element exists in the SMO

The following table shows how an attachment is created and sent if the SMO does not contain an `attachment` element with a `bodyPath` that matches a message name part:

Table 13. How the attachment is generated

Status of WSDL MIME binding for top-level binary message part	How message is created and sent
Present with one of the following: • No defined content type for the message part • Multiple content types defined • Wildcard content type defined	Message part is sent as an attachment. `Content-Id` is generated. `Content-Type` is set to `application/octet-stream`.
Present with single, non-wildcard content for the message part	Message part is sent as an attachment. `Content-Id` is generated. `Content-Type` is set to the type defined in the WSDL MIME content element.
Not present	Message part is not sent as an attachment.

Important: As described in "XML representation of SMO," the XSL Transformation mediation primitive transforms messages using an XSLT 1.0 transformation. The

transformation operates on an XML serialization of the SMO. The XSL Transformation mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Unreferenced attachments:

You can send and receive *unreferenced* attachments that are not declared in the service interface.

In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachments are in subsequent parts. No reference to the attachment is included in the SOAP body.

Figure 34. A SOAP message with an unreferenced attachment

You can send a SOAP message with an unreferenced attachment through a Web service export to a Web service import. The output message, which is sent to the target Web service, contains the attachment.

Figure 35. An attachment passing through an SCA module

In Figure 35, the SOAP message, with the attachment, passes through without modification.

You can also modify the SOAP message by using a mediation flow component. For example, you can use the mediation flow component to extract data from the SOAP message (binary data in the body of the message, in this case) and create a SOAP with attachments message. The data is processed as part of the attachments element of a service message object (SMO).

Figure 36. A message processed by a mediation flow component

Conversely, the mediation flow component can transform the incoming message by extracting and encoding the attachment and then transmitting the message with no attachments.

Instead of extracting data from an incoming SOAP message to form a SOAP with attachments message, you can obtain the attachment data from an external source, such as a database.

Figure 37. An attachment obtained from a database and added to the SOAP message

Conversely, the mediation flow component can extract the attachment from an incoming SOAP message and process the message (for example, store the attachment in a database).

Unreferenced attachments can be propagated only across mediation flow components. If an attachment must be accessed by or propagated to another component type, use a mediation flow component to move the attachment to a location that is accessible by that component.

Important: As described in "XML representation of SMO," the XSL Transformation mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The XSL Transformation mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Use of WSDL document style binding with multipart messages:

The Web Services Interoperability Organization (WS-I) organization has defined a set of rules regarding how Web services should be described by way of a WSDL and how the corresponding SOAP messages should be formed, in order to ensure interoperability.

These rules are specified in the WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html). In particular, the WS-I Basic Profile 1.1 R2712 states: "A document-literal binding MUST be serialized as an ENVELOPE with a

soap:Body whose child element is an instance of the global element declaration referenced by the corresponding wsdl:message part.”

This means that, when using a document style SOAP binding for an operation with messages (input, output, or fault) that are defined with multiple parts, only one of those parts should be bound to the SOAP body in order to be compliant with the WS-I Basic Profile 1.1.

Further, the WS-I Attachments Profile 1.0 R2941 states: “A wsdl:binding in a DESCRIPTION SHOULD bind every wsdl:part of a wsdl:message in the wsdl:portType to which it refers to one of soapbind:body, soapbind:header, soapbind:fault , soapbind:headerfault, or mime:content.”.

This means that, when using a document style SOAP binding for an operation with messages (input, output, or fault) that are defined with multiple parts, all parts other than the one selected to be bound to the SOAP body must be bound as attachments or headers.

The following approach is used when WSDL descriptions are generated for exports with Web service (JAX-WS and JAX-RPC) bindings in this case:

- You can choose which message part is bound to the SOAP body if there is more than one non-binary-typed element. If there is a single non-binary typed element, that element is automatically bound to the SOAP body.
- For the JAX-WS binding, all other message parts of type "hexBinary" or "base64Binary" are bound as referenced attachments. See “Referenced attachments: top-level message parts” on page 91.
- All other message parts are bound as SOAP headers.

The JAX-RPC and JAX-WS import bindings honor the SOAP binding in an existing WSDL document with multipart document style messages even if it does bind multiple parts to the SOAP body; however, you are not able to generate Web service clients for such WSDL documents in Rational Application Developer.

Note: The JAX-RPC binding does not support attachments.

The recommended pattern when using multipart messages with an operation that has document style SOAP binding is therefore:

1. Use the document/literal wrapped style. In this case, messages always have a single part; however, attachments have to be unreferenced (as described in “Unreferenced attachments” on page 97) or swaRef-typed (as described in “Referenced attachments: swaRef-typed elements” on page 87) in this case.
2. Use the RPC/literal style. In this case, there are no restrictions on the WSDL binding in terms of number of parts bound to the SOAP body; the SOAP message that results always has a single child that represents the operation being invoked, with the message parts being children of that element.
3. For the JAX-WS binding, have at most one message part that is not of type "hexBinary" or "base64Binary", unless it is acceptable to bind the other non-binary parts to SOAP headers.
4. Any other cases are subject to the behavior described.

Note: Additional restrictions exist when you use are using SOAP messages that do not conform to the WS-I *Basic Profile Version 1.1*.

- The first message part should be non-binary.

- When receiving multipart document-style SOAP messages with referenced attachments, the JAX-WS binding expects each referenced attachment to be represented by a SOAP body child element with an href attribute value that identifies the attachment by its content ID. The JAX-WS binding sends referenced attachments for such messages in the same way. This behavior is not compliant with the WS-I Basic Profile.

  To ensure that your messages comply with the Basic Profile, follow approach 1 on page 100 or 2 on page 100 in the previous list or avoid the use of referenced attachments for such messages and use unreferenced or swaRef-typed attachments instead.

HTTP bindings

The HTTP binding is designed to provide Service Component Architecture (SCA) connectivity to HTTP. Consequently, existing or newly-developed HTTP applications can participate in Service Oriented Architecture (SOA) environments.

Hypertext Transfer Protocol (HTTP) is a widely-used protocol for transferring information on the Web. When you are working with an external application that uses the HTTP protocol, an HTTP binding is necessary. The HTTP binding handles the transformation of data passed in as a message in a native format to a business object in an SCA application. The HTTP binding also can transform data passed out as a business object to the native format expected by the external application. for an incoming messaging.

Note: If you want to interact with clients and services that use the Web services SOAP/HTTP protocol, consider using one of the Web service bindings, which provide additional functionality with respect to handling Web services standard qualities of service.

Some common scenarios for using the HTTP binding are described in the following list:

- SCA-hosted services can invoke HTTP applications using an HTTP import.
- SCA-hosted services can expose themselves as HTTP-enabled applications, so they can be used by HTTP clients, using an HTTP export.
- WebSphere ESB and Process Server can communicate between themselves across an HTTP infrastructure, consequently users can manage their communications according to corporate standards.
- WebSphere ESB and Process Server can act as mediators of HTTP communications, transforming and routing messages, which improves the integration of applications using a HTTP network.
- WebSphere ESB and Process Server can be used to bridge between HTTP and other protocols, such as SOAP/HTTP Web services, Java Connector Architecture (JCA)-based resource adapters, JMS, and so on.

Detailed information about creating HTTP import and export bindings can be found in the Integration Designer information center. See the **Developing integration applications** > **Accessing external services with HTTP**> topics.

HTTP bindings overview:

The HTTP binding provides connectivity to HTTP-hosted applications. It mediates communication between HTTP applications and allows existing HTTP-based applications to be called from a module.

HTTP import bindings

The HTTP import binding provides outbound connectivity from Service Component Architecture (SCA) applications to an HTTP server or applications.

The import invokes an HTTP endpoint URL. The URL can be specified in one of three ways:

- The URL can be set dynamically in the HTTP headers by way of the dynamic override URL.
- The URL can be set dynamically in the SMO target address element.
- The URL can be specified as a configuration property on the import.

This invocation is always synchronous in nature.

Although HTTP invocations are always request-reply, the HTTP import supports both one-way and two-way operations and ignores the response in the case of a one-way operation.

HTTP export bindings

The HTTP export binding provides inbound connectivity from HTTP applications to an SCA application.

A URL is defined on the HTTP export. HTTP applications that want to send request messages to the export use this URL to invoke the export.

The HTTP export also supports pings.

HTTP bindings at runtime

An import with an HTTP binding at runtime sends a request with or without data in the body of the message from the SCA application to the external Web service. The request is made from the SCA application to the external Web service, as shown in Figure 38.

Figure 38. Flow of a request from the SCA application to the Web application

Optionally, the import with the HTTP binding can receive data back from the Web application in a response to the request.

With an export, the request is made by a client application to a Web service, as shown in Figure 39.

Figure 39. Flow of a request from the Web service to the client application.

The Web service is a Web application running on the server. The export is implemented in that Web application as a servlet so the client sends its request to a URL address. The servlet passes the request to the SCA application in the runtime.

Optionally, the export may send data to the client application in response to the request.

HTTP headers:

HTTP import and export bindings allow configuration of HTTP headers and their values to be used for outbound messages. The HTTP import uses these headers for requests, and the HTTP export uses them for responses.

Statically configured headers and control information take precedence over values dynamically set at runtime. However, the dynamic override URL, Version, and Method control values override the static values, which are otherwise considered defaults.

The binding supports the dynamic nature of the HTTP import URL by determining the value of HTTP target URL, Version, and Method at run time. These values are determined by extracting the value of Endpoint Reference, Dynamic Override URL, Version, and Method.

- For Endpoint Reference, use com.ibm.websphere.sca.addressing.EndpointReference APIs or set the /headers/SMOHeader/Target/address field in the SMO header.
- For Dynamic Override URL, Version, and Method, use the HTTP control parameters section of the Service Component Architecture (SCA) message. Note that the Dynamic Override URL takes precedence over the target Endpoint Reference; however, the Endpoint Reference applies across bindings, so it is the preferred approach and should be used where possible.

The control and header information for outbound messages under HTTP export and import bindings is processed in the following order:

1. Header and control information excluding HTTP dynamic override URL, Version, and Method from the SCA Message (lowest priority)
2. Changes from the administrative console on the export/import level
3. Changes from the administrative console on the method level of the export or import
4. Target address specified by way of the Endpoint Reference or the SMO header
5. Dynamic Override URL, Version, and Method from the SCA message
6. Headers and control information from the data handler or data binding (highest priority)

The HTTP export and import will populate inbound direction headers and control parameters with data from the incoming message (HTTPExportRequest and HTTPImportResponse) only if protocol header propagation is set to `True`. Inversely, the HTTP export and import will read and process outbound headers and control parameters (HTTPExportResponse and HTTPImportRequest) only if protocol header propagation is set to `True`.

Note: Data handler or data binding changes to headers or control parameters in the import response or export request will not alter the processing instructions of the message inside the import or export binding and should be used only to propagate modified values to downstream SCA components.

The context service is responsible for propagating the context (including the protocol headers, such as the HTTP header, and the user context, such as account ID) along an SCA invocation path. During development in IBM Integration Designer, you can control the propagation of context by way of import and export properties. For more details, see the import and export bindings information in the IBM Integration Designer information center.

Supplied HTTP header structures and support

Table 14 itemizes the request/response parameters for HTTP Import and HTTP Export requests and responses.

Table 14. Supplied HTTP header information

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
URL	Ignored	Not set	Read from the request message. **Note:** Query string is also part of the URL control parameter.	Ignored
Version (possible values: 1.0, 1.1; default is 1.1)	Ignored	Not set	Read from the request message	Ignored
Method	Ignored	Not set	Read from the request message	Ignored
Dynamic Override URL	If set in the data handler or data binding, overrides the HTTP Import URL. Written to the message in the request line. **Note:** Query string is also part of the URL control parameter.	Not set	Not set	Ignored
Dynamic Override Version	If set, overrides the HTTP Import Version. Written to the message in the request line.	Not set	Not set	Ignored
Dynamic Override Method	If set, overrides the HTTP Import Method. Written to the message in the request line.	Not set	Not set	Ignored
Media Type (This control parameter carries part of the value of the Content-Type HTTP header.)	If present, written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data handler or data binding.	Read from the response message, Content-Type header	Read from the request message, Content-Type header	If present, written to the message as part of Content-Type header. **Note:** This control element value should be provided by the data handler or data binding.

Table 14. Supplied HTTP header information (continued)

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
Character set (default: UTF-8)	If present, written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data binding.	Read from the response message, Content-Type header	Read from the request message, Content-Type header	Supported; written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data binding.
Transfer Encoding (Possible values: chunked, identity; default is identity)	If present, written to the message as a header and controls how the message transformation is encoded.	Read from the response message	Read from the request message	If present, written to the message as a header and controls how the message transformation is encoded.
Content Encoding (Possible values: gzip, x-gzip, deflate, identity; default is identity)	If present, written to the message as a header and controls how the payload is encoded.	Read from the response message	Read from the request message	If present, written to the message as a header and controls how the payload is encoded.
Content-Length	Ignored	Read from the response message	Read from the request message	Ignored
StatusCode (default: 200)	Not supported	Read from the response message	Not supported	If present, written to the message in the response line
ReasonPhrase (default: OK)	Not supported	Read from the response message	Not supported	Control value ignored. The message response line value is generated from the StatusCode.
Authentication (contains multiple properties)	If present, used to construct the Basic Authentication header. **Note:** The value for this header will be encoded only on the HTTP protocol. In the SCA, it will be decoded and passed as clear text.	Not applicable	Read from the request message Basic Authentication header. The presence of this header does not indicate the user has been authenticated. Authentication should be controlled in the servlet configuration. **Note:** The value for this header will be encoded only on the HTTP protocol. In the SCA, it will be decoded and passed as clear text.	Not applicable
Proxy (contains multiple properties: Host, Port, Authentication)	If present, used to establish connection through proxy.	Not applicable	Not applicable	Not applicable

Table 14. Supplied HTTP header information (continued)

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
SSL (contains multiple properties: Keystore, Keystore Password, Trustore, Trustore Password, ClientAuth)	If populated and the destination url is HTTPS, it is used to establish a connection through SSL.	Not applicable	Not applicable	Not applicable

HTTP data bindings:

For each different mapping of data between a Service Component Architecture (SCA) message and an HTTP protocol message, a data handler or an HTTP data binding must be configured. Data handlers provide a binding-neutral interface that allows reuse across transport bindings and represent the recommended approach; data bindings are specific to a particular transport binding. HTTP-specific data binding classes are supplied; you can also write custom data handlers or data bindings.

Note: The three HTTP data binding classes described in this topic (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML, and HTTPServiceGatewayDataBinding) are deprecated as of WebSphere ESB Version 7.0. Instead of using the data bindings described in this topic, consider the following data handlers:

- Use SOAPDataHandler instead of HTTPStreamDataBindingSOAP.
- Use UTF8XMLDataHandler instead of HTTPStreamDataBindingXML
- Use GatewayTextDataHandler instead of HTTPServiceGatewayDataBinding

Data bindings are provided for use with HTTP imports and HTTP exports: binary data binding, XML data binding, and SOAP data binding. A response data binding is not required for one-way operations. A data binding is represented by the name of a Java class whose instances can convert both from HTTP to ServiceDataObject and vice-versa. A function selector must be used on an export which, in conjunction with method bindings, can determine which data binding is used and which operation is invoked. The supplied data bindings are:

- Binary data bindings, which treat the body as unstructured binary data. The implementation of the binary data binding XSD schema is as follows:

```
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:tns="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="HTTPBaseBody">
    <xsd:sequence/>
  </xsd:complexType>

  <xsd:complexType name="HTTPBytesBody">
    <xsd:complexContent>
      <xsd:extension base="tns:HTTPBaseBody">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:hexBinary"/>
```

```
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

- XML data bindings, which support the body as XML data. The implementation of the XML data binding is similar to the JMS XML data binding and has no restrictions on the interface schema.
- SOAP data bindings, which support the body as SOAP data. The implementation of the SOAP data binding has no restrictions on the interface schema.

Implementing custom HTTP data bindings

This section describes how to implement a custom HTTP data binding.

Note: The recommended approach is to implement a custom data handler because it can be reused across transport bindings.

HTTPStreamDataBinding is the principal interface for handling custom HTTP messages. The interface is designed to allow handling of large payloads. However, in order for such implementation to work, this data binding must return the control information and headers before writing the message into the stream.

The methods and their order of execution, listed below, must be implemented by the custom data binding.

To customize a data binding, write a class that implements HTTPStreamDataBinding. The data binding should have four private properties:

- private DataObject pDataObject
- private HTTPControl pCtrl
- private HTTPHeaders pHeaders
- private yourNativeDataType nativeData

The HTTP binding will invoke the customized data binding in the following order:

- Outbound processing (DataObject to Native format):
  1. setDataObject(...)
  2. setHeaders(...)
  3. setControlParameters(...)
  4. setBusinessException(...)
  5. convertToNativeData()
  6. getControlParameters()
  7. getHeaders()
  8. write(...)
- Inbound processing (Native format to DataObject):
  1. setControlParameters(...)
  2. setHeaders(...)
  3. convertFromNativeData(...)
  4. isBusinessException()
  5. getDataObject()
  6. getControlParameters()

7. getHeaders()

You need to invoke setDataObject(...) in convertFromNativeData(...) to set the value of dataObject, which is converted from native data to the private property "pDataObject".

```
public void setDataObject(DataObject dataObject)
   throws DataBindingException {
  pDataObject = dataObject;

}
public void setControlParameters(HTTPControl arg0) {
  this.pCtrl = arg0;
}

public void setHeaders(HTTPHeaders arg0) {
  this.pHeaders = arg0;
}
/*
* Add http header "IsBusinessException" in pHeaders.
* Two steps:
* 1.Remove all the header with name IsBusinessException (case-insensitive) first.
*   This is to make sure only one header is present.
* 2.Add the new header "IsBusinessException"
*/
public void setBusinessException(boolean isBusinessException) {
  //remove all the header with name IsBusinessException (case-insensitive) first.
  //This is to make sure only one header is present.
  //add the new header "IsBusinessException", code example:
  HTTPHeader header=HeadersFactory.eINSTANCE.createHTTPHeader();
  header.setName("IsBusinessException");
  header.setValue(Boolean.toString(isBusinessException));
  this.pHeaders.getHeader().add(header);

}
public HTTPControl getControlParameters() {
  return pCtrl;
}
public HTTPHeaders getHeaders() {
  return pHeaders;
}
public DataObject getDataObject() throws DataBindingException {
  return pDataObject;
}
/*
* Get header "IsBusinessException" from pHeaders, return its boolean value
*/
public boolean isBusinessException() {
  String headerValue = getHeaderValue(pHeaders,"IsBusinessException");
  boolean result=Boolean.parseBoolean(headerValue);
  return result;
}
public void convertToNativeData() throws DataBindingException {
  DataObject dataObject = getDataObject();
  this.nativeData=realConvertWorkFromSDOToNativeData(dataObject);
}
public void convertFromNativeData(HTTPInputStream arg0){
  //Customer-developed method to
  //Read data from HTTPInputStream
  //Convert it to DataObject
  DataObject dataobject=realConvertWorkFromNativeDataToSDO(arg0);
  setDataObject(dataobject);
}
```

```
public void write(HTTPOutputStream output) throws IOException {
  if (nativeData != null)
  output.write(nativeData);
}
```

EJB bindings

Enterprise JavaBeans (EJB) import bindings enable Service Component Architecture (SCA) components to invoke services provided by Java EE business logic running on a Java EE server. EJB export bindings allow SCA components to be exposed as Enterprise JavaBeans so that Java EE business logic can invoke SCA components otherwise unavailable to them.

EJB import bindings:

EJB import bindings allow an SCA module to call EJB implementations by specifying the way that the consuming module is bound to the external EJB. Importing services from an external EJB implementation allows users to plug their business logic into the WebSphere ESB environment and participate in a business process.

You use Integration Designer to create EJB import bindings. You can use either of the following procedures to generate the bindings:

- Creating EJB import using the external service wizard

  You can use the external service wizard in Integration Designer to build an EJB import based on an existing implementation. The external service wizard creates services based on criteria that you provide. It then generates business objects, interfaces, and import files based on the services discovered.
- Creating EJB import using the assembly editor

  You can create an EJB import within an assembly diagram using the Integration Designer assembly editor. From the palette, you can use either an Import or use a Java class to create the EJB binding.

The generated import has data bindings that make the Java-WSDL connection instead of requiring a Java bridge component. You can directly wire a component with a Web Services Description Language (WSDL) reference to the EJB import that communicates to an EJB-based service using a Java interface.

The EJB import can interact with Java EE business logic using either the EJB 2.1 programming model or the EJB 3.0 programming model.

The invocation to the Java EE business logic can be local (for EJB 3.0 only) or remote.

- Local invocation is used when you want to call Java EE business logic that resides on the same server as the import.

Figure 40. Local invocation of an EJB (EJB 3.0 only)

- Remote invocation is used when you want to call Java EE business logic that does not reside on the same server as the import.

  For example, in the following figure, an EJB import uses the Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP) to invoke an EJB method on another server.

Figure 41. Remote invocation of an EJB

When it configures the EJB binding, Integration Designer uses the JNDI name to determine the EJB programming model level and the type of invocation (local or remote).

EJB import bindings contain the following major components:

- JAX-WS data handler
- EJB fault selector
- EJB import function selector

If your user scenario is not based on the JAX-WS mapping, you might need a custom data handler, function selector, and fault selector to perform the tasks otherwise completed by the components that are part of the EJB import bindings. This includes the mapping normally completed by the custom mapping algorithm.

EJB export bindings:

External Java EE applications can invoke an SCA component by way of an EJB export binding. Using an EJB export lets you expose SCA components so that external Java EE applications can invoke those components using the EJB programming model.

Note: The EJB export is a stateless bean.

You use Integration Designer to create EJB bindings. You can use either of the following procedures to generate the bindings:

- Creating EJB export bindings using the external service wizard

  You can use the external service wizard in Integration Designer to build an EJB export service based on an existing implementation. The external service wizard creates services based on criteria that you provide. It then generates business objects, interfaces, and export files based on the services discovered.
- Creating EJB export bindings using the assembly editor

  You can create an EJB export using the Integration Designer assembly editor.

Important: A Java 2 Platform, Standard Edition (J2SE) client cannot invoke the EJB export client that is generated in Integration Designer.

You can generate the binding from an existing SCA component, or you can generate an export with an EJB binding for a Java interface.

- When you generate an export for an existing SCA component that has an existing WSDL interface, the export is assigned a Java interface.
- When you generate an export for a Java interface, you can select either a WSDL or a Java interface for the export.

  Note: A Java interface used to create an EJB export has the following limitations with regard to the objects (input and output parameters and exceptions) passed as parameters on a remote call:
  - They must be of concrete type (instead of an interface or abstract type).
  - They must conform to the Enterprise JavaBeans specification. They must be serializable and have the default no-argument constructor, and all properties must be accessible through getter and setter methods.

    Refer to the Sun Microsystems, Inc., Web site at http://java.sun.com for information about the Enterprise JavaBeans specification.

  In addition, the exception must be a checked exception, inherited from java.lang.Exception, and it must be singular (that is, it does not support throwing multiple checked exception types).

  Note also that the business interface of a Java EnterpriseBean is a plain Java interface and must not extend javax.ejb.EJBObject or javax.ejb.EJBLocalObject. The methods of the business interface should not throw java.rmi.Remote.Exception.

The EJB export bindings can interact with Java EE business logic using either the EJB 2.1 programming model or the EJB 3.0 programming model.

The invocation can be local (for EJB 3.0 only) or remote.

- Local invocation is used when the Java EE business logic calls an SCA component that resides on the same server as the export.
- Remote invocation is used when the Java EE business logic does not reside on the same server as the export.

  For example, in the following figure, an EJB uses RMI/IIOP to call an SCA component on a different server.

Figure 42. Remote call from a client to an SCA component by way of an EJB export

When it configures the EJB binding, Integration Designer uses the JNDI name to determine the EJB programming model level and the type of invocation (local or remote).

EJB export bindings contain the following major components:

- JAX-WS data handler
- EJB export function selector

If your user scenario is not based on the JAX-WS mapping, you might need a custom data handler and function selector to perform the tasks otherwise completed by the components that are part of the EJB export bindings. This includes the mapping normally completed by the custom mapping algorithm.

EJB binding properties:

EJB import bindings use their configured JNDI names to determine the EJB programming model level and type of invocation (local or remote). EJB import and export bindings use the JAX-WS data handler for data transformation. The EJB import binding uses an EJB import function selector and an EJB fault selector, and the EJB export binding uses an EJB export function selector.

JNDI names and EJB import bindings:

When it configures the EJB binding on an import, Integration Designer uses the JNDI name to determine the EJB programming model level and type of invocation (local or remote).

If no JNDI name is specified, the default EJB interface binding is used. The default names that are created depend on whether you are invoking EJB 2.1 JavaBeans or EJB 3.0 JavaBeans.

Note: Refer to the WebSphere Application Server information center for more detailed information about naming conventions: EJB 3.0 application bindings overview.

- EJB 2.1 JavaBeans

  The default JNDI name preselected by Integration Designer is the default EJB 2.1 binding, which takes the form `ejb/` plus the home interface, separated by slashes.

  For example, for the home interface of EJB 2.1 JavaBeans for com.mycompany.myremotebusinesshome, the default binding is:

  ```
  ejb/com/mycompany/myremotebusinesshome
  ```

  For EJB 2.1, only remote EJB invocation is supported.

- EJB 3.0 JavaBeans

  The default JNDI name preselected by Integration Designer for the local JNDI is the fully qualified class name of the local interface preceded by `ejblocal:`. For example, for the fully qualified interface of the local interface com.mycompany.mylocalbusiness, the preselected EJB 3.0 JNDI is:

  ```
  ejblocal:com.mycompany.mylocalbusiness
  ```

  For the remote interface com.mycompany.myremotebusiness, the preselected EJB 3.0 JNDI is the fully qualified interface:

  ```
  com.mycompany.myremotebusiness
  ```

  The EJB 3.0 default application bindings are described at the following location: EJB 3.0 application bindings overview.

  Integration Designer will use the "short" name as the default JNDI location for EJBs using the version 3.0 programming model.

  Note: If the deployed JNDI reference of the target EJB is different from the default JNDI binding location because a custom mapping was used or configured, the target JNDI name must be properly specified. You can specify

the name in Integration Designer before deployment, or, for the import binding, you can change the name in the administrative console (after deployment) to match the JNDI name of the target EJB.

For more information on creating EJB bindings, see the section devoted to Working with EJB bindings in the Integration Designer information center.

JAX-WS data handler:

The Enterprise JavaBeans (EJB) import binding uses the JAX-WS data handler to turn request business objects into Java object parameters and to turn the Java object return value into the response business object. The EJB export binding uses the JAX-WS data handler to turn request EJBs into request business objects and to turn the response business object into a return value.

This data handler maps data from the SCA-specified WSDL interface to the target EJB Java interface (and vice versa) using the Java API for XML Web Services (JAX-WS) specification and the Java Architecture for XML Binding (JAXB) specification.

Note: Current support is restricted to the JAX-WS 2.1.1 and JAXB 2.1.3 specifications.

The data handler specified at the EJB binding level is used to perform request, response, fault, and runtime exception processing.

Note: For faults, a specific data handler can be specified for each fault by specifying the faultBindingType configuration property. This overrides the value specified at the EJB binding level.

The JAX-WS data handler is used by default when the EJB binding has a WSDL interface. This data handler cannot be used to transform a SOAP message representing a JAX-WS invocation to a data object.

The EJB import binding uses a data handler to transform a data object into a Java Object array (Object[]). During outbound communications, the following processing takes place:

1. The EJB binding sets the expected type, expected element, and targeted method name in the BindingContext to match those specified in the WSDL.
2. The EJB binding invokes the transform method for the data object requiring data transformation.
3. The data handler returns an Object[] representing the parameters of the method (in the order of their definition within the method).
4. The EJB binding uses the Object[] to invoke the method on the target EJB interface.

The binding also prepares an Object[] to process the response from the EJB invocation.

- The first element in the Object[] is the return value from the Java method invocation.
- The subsequent values represent the input parameters for the method.

This is required to support the In/Out and Out types of parameters.

For parameters of type Out, the values must be returned in the response data object.

The data handler processes and transforms values found in the Object[] and then returns a response to the data object.

The data handler supports xs:AnyType, xs:AnySimpleType, and xs:Any along with other XSD data types. To enable support for xs:Any, use the `@XmlAnyElement (lax=true)` for the JavaBeans property in the Java code, as shown in the following example:

```
public class TestType {
 private Object[] object;

 @XmlAnyElement (lax=true)
 public Object[] getObject() {
  return object;
 )

 public void setObject (Object[] object) {
  this.object=object;
 )
)
```

This makes the property object in TestType an xs:any field. The Java class value used in the xs:any field should have the `@XmlAnyElement` annotation. For example, if Address is the Java class being used to populate the object array, the Address class should have the annotation `@XmlRootElement`.

Note: To customize the mapping from the XSD type to Java types defined by the JAX-WS specification, change the JAXB annotations to fit your business need. The JAX-WS data handler supports xs:any, xs:anyType, and xs:anySimpleType.

The following restrictions are applicable for the JAX-WS data handler:

- The data handler does not include support for the header attribute `@WebParam` annotation.
- The namespace for business object schema files (XSD files) does not include default mapping from the Java package name. The annotation `@XMLSchema` in package-info.java also does not work. The only way to create an XSD with a namespace is to use the `@XmlType` and `@XmlRootElement` annotations. `@XmlRootElement` defines the target namespace for the global element in JavaBeans types.
- The EJB import wizard does not create XSD files for unrelated classes. Version 2.0 does not support the `@XmlSeeAlso` annotation, so if the child class is not referenced directly from the parent class, an XSD is not created. The solution to this problem is to run SchemaGen for such child classes.

  SchemaGen is a command line utility (located in the *WPS_Install_Home*/bin directory) provided to create XSD files for a given bean. These XSDs must be manually copied to the module for the solution to work.

EJB fault selector:

The EJB fault selector determines if an EJB invocation has resulted in a fault, a runtime exception, or a successful response.

If a fault is detected, the EJB fault selector returns the native fault name to the binding runtime so the JAX-WS data handler can convert the exception object into a fault business object.

On a successful (non-fault) response, the EJB import binding assembles a Java object array (Object[]) to return the values.

- The first element in the Object[] is the return value from the Java method invocation.
- The subsequent values represent the input parameters for the method.

This is required to support the In/Out and Out types of parameters.

For exception scenarios, the binding assembles an Object[] and the first element represents the exception thrown by the method.

The fault selector can return any of the following values:

Table 15. Return values

Type	Return value	Description
Fault	`ResponseType.FAULT`	Returned when the passed Object[] contains an exception object.
Runtime exception	`ResponseType.RUNTIME`	Returned if the exception object does not match any of the declared exception types on the method.
Normal response	`ResponseType.RESPONSE`	Returned in all other cases.

If the fault selector returns a value of `ResponseType.FAULT`, the native fault name is returned. This native fault name is used by the binding to determine the corresponding WSDL fault name from the model and to invoke the correct fault data handler.

EJB function selector:

The EJB bindings use an import function selector (for outbound processing) or an export function selector (for inbound processing) to determine the EJB method to call.

Import function selector

For outbound processing, the import function selector derives the EJB method type based on the name of the operation invoked by the SCA component that is wired to the EJB import. The function selector looks for the @WebMethod annotation on the Integration Designer-generated JAX-WS annotated Java class to determine the associated target operation name.

- If the @WebMethod annotation is present, the function selector uses the @WebMethod annotation to determine the correct Java method mapping for the WSDL method.
- If the @WebMethod annotation is missing, the function selector assumes that the Java method name is the same as the invoked operation name.

Note: This function selector is valid only for a WSDL-typed interface on an EJB import, not for a Java-typed interface on an EJB import.

The function selector returns a java.lang.reflect.Method object that represents the method of the EJB interface.

The function selector uses a Java Object array (Object[]) to contain the response from the target method. The first element in the Object[] is a Java method with the name of the WSDL, and the second element in the Object[] is the input business object.

Export function selector

For inbound processing, the export function selector derives the target method to be invoked from the Java method.

The export function selector maps the Java operation name invoked by the EJB client into the name of the operation in the interface of the target component. The method name is returned as a String and is resolved by the SCA runtime depending on the interface type of the target component.

EIS bindings

Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved using EIS exports and EIS imports that support JCA 1.5 resource adapters and Websphere Adapters.

Your SCA components might require that data be transferred to or from an external EIS. When you create an SCA module requiring such connectivity, you will include (in addition to your SCA component) an import or export with an EIS binding for communication with a specific external EIS.

Resource adapters in IBM Integration Designer are used within the context of an import or an export. You develop an import or an export with the external service wizard and, in developing it, include the resource adapter. An EIS import, which lets your application invoke a service on an EIS system, or an EIS export, which lets an application on an EIS system invoke a service developed in IBM Integration Designer, are created with a resource adapter. For example, you would create an import with the JD Edwards adapter to invoke a service on the JD Edwards system.

When you use the external service wizard, the EIS binding information is created for you. You can also use another tool, the assembly editor, to add or modify binding information. See the Accessing external services with adapters for more information.

After the SCA module containing the EIS binding is deployed to the server, you can use the administrative console to view information about the binding or to configure the binding.

EIS bindings overview:

The EIS (enterprise information system) binding, when used with a JCA resource adapter, lets you access services on an enterprise information system or make your services available to the EIS.

The following example shows how an SCA module named ContactSyncModule synchronizes contact information between a Siebel system and an SAP system.

1. The SCA component named ContactSync listens (by way of an EIS application export named Siebel Contact) for changes to Siebel contacts.
2. The ContactSync SCA component itself makes use of an SAP application (through an EIS application import) in order to update the SAP contact information accordingly.

Because the data structures used for storing contacts are different in Siebel and SAP systems, the ContactSync SCA component must provide mapping.

Figure 43. Flow from a Siebel system to an SAP system

The Siebel Contact export and the SAP Contact import have the appropriate resource adapters configured.

Key features of EIS bindings:

An EIS import is a Service Component Architecture (SCA) import that allows components in the SCA module to use EIS applications defined outside the SCA module. An EIS import is used to transfer data from the SCA component to an external EIS; an EIS export is used to transfer data from an external EIS into the SCA module.

Imports

The role of the EIS import is to bridge the gap between SCA components and external EIS systems. External applications can be treated as an EIS import. In this case, the EIS import sends data to the external EIS and optionally receives data in response.

The EIS import provides SCA components with a uniform view of the applications external to the module. This allows components to communicate with an external EIS, such as SAP, Siebel, or PeopleSoft, using a consistent SCA model.

On the client side of the import, there is an interface, exposed by the EIS import application, with one or more methods, each taking data objects as arguments and return values. On the implementation side, there is a Common Client Interface (CCI) implemented by a resource adapter.

The runtime implementation of an EIS import connects the client-side interface and the CCI. The import maps the invocation of the method on the interface to the invocation on the CCI.

Bindings are created at three levels: the interface binding, which then uses the contained method bindings, which in turn use data bindings.

The interface binding relates the interface of the import to the connection to the EIS system providing the application. This reflects the fact that the set of applications, represented by the interface, is provided by the specific instance of the EIS and the connection provides access to this instance. The binding element contains properties with enough information to create the connection (these properties are part of the javax.resource.spi.ManagedConnectionFactory instance).

The method binding associates the method with the specific interaction with the EIS system. For JCA, the interaction is characterized by the set of properties of the javax.resource.cci.InteractionSpec interface implementation. The interaction element of the method binding contains these properties, along with the name of the class, thus providing enough information to perform the interaction. The method binding uses data bindings describing the mapping of the argument and result of the interface method to EIS representation.

The runtime scenario for an EIS import is as follows:

1. The method on the import interface is invoked using the SCA programming model.
2. The request, reaching the EIS import, contains the name of the method and its arguments.
3. The import first creates an interface binding implementation; then, using data from the import binding, it creates a ConnectionFactory and associates the two. That is, the import calls setConnectionFactory on the interface binding.
4. The method binding implementation matching the invoked method is created.
5. The javax.resource.cci.InteractionSpec instance is created and populated; then, data bindings are used to bind the method arguments to a format understood by the resource adapter.
6. The CCI interface is used to perform the interaction.
7. When the call returns, the data binding is used to create the result of the invocation, and the result is returned to the caller.

Exports

The role of the EIS export is to bridge the gap between an SCA component and an external EIS. External applications can be treated as an EIS export. In this case, the external application sends its data in the form of periodic notifications. An EIS export can be thought of as a subscription application listening to an external request from an EIS. The SCA component that uses the EIS export views it as a local application.

The EIS export provides SCA components with a uniform view of the applications external to the module. This allows components to communicate with an EIS, such as SAP, Siebel, or PeopleSoft, using a consistent SCA model.

The export features a listener implementation receiving requests from the EIS. The listener implements a resource adapter-specific listener interface. The export also contains a component implementing interface, exposed to the EIS through the export.

The runtime implementation of an EIS export connects the listener with the component implementing interface. The export maps the EIS request to the invocation of the appropriate operation on the component. Bindings are created at three levels: a listener binding, which then uses a contained native method binding, which in turn uses a data binding .

The listener binding relates the listener receiving requests with the component exposed through the export. The export definition contains the name of the component; the runtime locates it and forwards requests to it.

The native method binding associates the native method or the event type received by the listener to the operation implemented by the component exposed by way of the export. There is no relationship between the method invoked on the listener and the event type; all the events arrive through one or more methods of the listener. The native method binding uses the function selector defined in the export to extract the native method name from the inbound data and data bindings to bind the data format of the EIS to a format understood by the component.

The runtime scenario for an EIS export is as follows:

1. The EIS request triggers invocation of the method on the listener implementation.
2. The listener locates and invokes the export, passing to it all the invocation arguments.
3. The export creates the listener binding implementation.
4. The export instantiates the function selector and sets it on the listener binding.
5. The export initializes native method bindings and adds them to the listener binding. For each native method binding, the data bindings are also initialized.
6. The export invokes the listener binding.
7. The listener binding locates exported components and uses the function selector to retrieve the native method name.
8. This name is used to locate the native method binding, which then invokes the target component.

The adapter interaction style allows for the EIS export binding to invoke the target component either asynchronously (the default) or synchronously.

Resource adapters

You develop an import or an export with the external service wizard and, in developing it, include a resource adapter. The adapters that come with IBM Integration Designer used to access CICS, IMS, JD Edwards, PeopleSoft, SAP and Siebel systems are intended for development and test purposes only. This means you use them to develop and test your applications.

Once you deploy your application, you will need licensed runtime adapters to run your application. However, when you build your service you can embed the adapter with your service. Your adapter licensing might allow you to use the embedded adapter as the licensed runtime adapter. These adapters are compliant with the Java EE Connector Architecture (JCA 1.5). JCA, an open standard, is the Java EE standard for EIS connectivity. JCA provides a managed framework; that is, Quality of Service (QoS) is provided by the application server, which offers life-cycle management and security to transactions. They are also compliant with the Enterprise Metadata Discovery specification with the exception of IBM CICS ECI Resource Adapter and IBM IMS Connector for Java.

The WebSphere Business Integration Adapters, an older set of adapters, are also supported by the wizard.

Java EE resources

The EIS module, an SCA module that follows the EIS module pattern, can be deployed to the Java EE platform.

The deployment of the EIS module to the Java EE platform results in an application that is ready to execute, packaged as an EAR file and deployed to the server. All Java EE artifacts and resources are created; the application is configured and ready to be run.

JCA Interaction Spec and Connection Spec dynamic properties:

The EIS binding can accept input for the InteractionSpec and ConnectionSpec specified by using a well-defined child data object that accompanies the payload. This allows for dynamic request-response interactions with a resource adapter through the InteractionSpec and component authentication through the ConnectionSpec.

The javax.cci.InteractionSpec carries information on how the interaction request with the resource adapter should be handled. It can also carry information on how the interaction was achieved after the request. These two-way communications through the interactions are sometimes referred to as *conversations*.

The EIS binding expects the payload that will be the argument to the resource adapter to contain a child data object called `properties`. This property data object will contain name/value pairs, with the name of the Interaction Spec properties in a specific format. The formatting rules are:

- Names must begin with the prefix `IS`, followed by the property name. For example, an interaction spec with a JavaBeans property called `InteractionId` would specify the property name as `ISInteractionId`.
- The name/value pair represents the name and the value of the simple type of the Interaction Spec property.

In this example, an interface specifies that the input of an operation is an `Account` data object. This interface invokes an EIS import binding application with the intention to send and receive a dynamic InteractionSpec property called `workingSet` with the value `xyz`.

The business graph or business objects in the server contain an underlying `properties` business object that permits the sending of protocol-specific data with the payload. This `properties` business object is built-in and does not need to be specified in the XML schema when constructing a business object. It only needs to be created and used. If you have your own data types defined based on an XML schema, you need to specify a `properties` element that contains your expected name/value pairs.

```
BOFactory dataFactory = (BOFactory) \
serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
 //Wrapper for doc-lit wrapped style interfaces,
 //skip to payload for non doc-lit
 DataObject docLitWrapper = dataFactory.createByElement /
 ("http://mytest/eis/Account", "AccountWrapper");
```

Create the payload.

```
DataObject account = docLitWrapper.createDataObject(0);
  DataObject accountInfo = account.createDataObject("AccountInfo");
  //Perform your setting up of payload
```

```
//Construct properties data for dynamic interaction

DataObject properties = account.createDataObject("properties");
```

For name workingSet, set the value expected (xyz).

```
properties.setString("ISworkingSet", "xyz");
```

```
//Invoke the service with argument

Service accountImport = (Service) \
serviceManager.locateService("AccountOutbound");
DataObject result = accountImport.invoke("createAccount", docLitWrapper);

//Get returned property
DataObject retProperties = result.getDataObject("properties");

String workingset = retProperties.getString("ISworkingSet");
```

You can use ConnectionSpec properties for dynamic component authentication. The same rules apply as above, except that the property name prefix needs to be CS (instead of IS). ConnectionSpec properties are not two-way. The same properties data object can contain both IS and CS properties.

To use ConnectionSpec properties, set the resAuth specified on the import binding to Application. Also, make sure the resource adapter supports component authorization. See chapter 8 of the J2EE Connector Architecture Specification for more details.

External clients with EIS bindings:

The server can send messages to, or receive messages from, external clients using EIS bindings.

An external client, for example a Web portal or an EIS, needs to send a message to an SCA module in the server or needs to be invoked by a component from within the server.

The client invokes the EIS import as with any other application, using either the Dynamic Invocation Interface (DII) or Java interface.

1. The external client creates an instance of the ServiceManager and looks up the EIS import using its reference name. The result of the lookup is a service interface implementation.
2. The client creates an input argument, a generic data object, created dynamically using the data object schema. This step is done using the Service Data Object DataFactory interface implementation.
3. The external client invokes the EIS and obtains the required results.

Alternatively, the client can invoke the EIS import using the Java interface.

1. The client creates an instance of the ServiceManager and looks up the EIS import using its reference name. The result of the lookup is a Java interface of the EIS import.
2. The client creates an input argument and a typed data object.
3. The client invokes EIS and obtains the required results.

The EIS export interface defines the interface of the exported SCA component that is available to the external EIS applications. This interface can be thought of as the interface that an external application (such as SAP or PeopleSoft) will invoke through the implementation of the EIS export application runtime.

The export uses EISExportBinding to bind exported services to the external EIS application. It allows you to subscribe an application contained in your SCA module to listen for EIS service requests. The EIS export binding specifies the mapping between the definition of inbound events as it is understood by the resource adapter (using Java EE Connector Architecture interfaces) and the invocation of SCA operations.

The EISExportBinding requires external EIS services to be based on Java EE Connector Architecture 1.5 inbound contracts. The EISExportBinding requires that a data handler or data binding be specified either at the binding level or the method level.

JMS bindings

A Java Message Service (JMS) provider enables messaging based on the Java Messaging Service API and programming model. It provides Java EE connection factories to create connections for JMS destinations and to send and receive messages.

Three JMS bindings are provided:

- Service integration bus (SIB) provider binding compliant with JMS JCA 1.5 (*JMS binding*)
- Generic, non-JCA JMS binding, compliant with JMS 1.1 (*Generic JMS binding*)
- WebSphere MQ JMS binding, providing JMS provider support for WebSphere MQ and allowing Java EE application interoperability (*WebSphere MQ JMS binding*)

The JMS export and import bindings allow a Service Component Architecture (SCA) module to make calls to, and receive messages from, external JMS systems.

Also supported are WebSphere MQ bindings (*WebSphere MQ binding*) which allow native MQ users to handle arbitrary incoming and outgoing message formats (WebSphere MQ required).

The JMS import and export bindings provide integration with JMS applications using the JCA 1.5-based SIB JMS provider that is part of WebSphere Application Server. Other JCA 1.5–based JMS resource adapters are not supported

JMS bindings overview:

JMS bindings provide connectivity between the Service Component Architecture (SCA) environment and JMS systems.

JMS bindings

The major components of both JMS import and JMS export bindings are:

- Resource adapter: enables managed, bidirectional connectivity between an SCA module and external JMS systems
- Connections: encapsulate a virtual connection between a client and a provider application

- Destinations: used by a client to specify the target of messages it produces or the source of messages it consumes
- Authentication data: used to secure access to the binding

JMS import bindings

JMS import bindings allow you to import an external JMS application to be used inside your SCA module. JMS import bindings allow components within your SCA module to communicate with services provided by external JMS applications.

Connections to the associated JMS provider of JMS destinations are created by using a JMS connection factory. Use connection factory administrative objects to manage JMS connection factories for the default messaging provider.

Interaction with external JMS systems includes the use of destinations for sending requests and receiving replies.

Two types of usage scenarios for the JMS import binding are supported, depending on the type of operation being invoked:

- One-way: The JMS import puts a message on the send destination configured in the import binding. Nothing is set in the replyTo field of the JMS header.
- Two-way (request-response): The JMS import puts a message on the send destination and then persists the reply it receives from the SCA component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default), or from the request message correlation ID. The import binding can also be configured to use a temporary dynamic response destination to correlate responses with requests. A temporary destination is created for each request and the import uses this destination to receive the response.

  The receive destination is set in the replyTo header property of the outbound message. A message listener is deployed to listen on the receive destination, and when a reply is received, the message listener passes the reply back to the component.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified. Static properties can be set from the JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that JMS is an asynchronous binding. If a calling component invokes a JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 44 on page 124 illustrates how the import is linked to the external service.

Figure 44. JMS import binding resources

JMS export bindings

JMS export bindings provide the means for SCA modules to provide services to external JMS applications.

The connection that is part of a JMS export is a configurable activation specification.

A JMS export has send and receive destinations.

- The receive destination is where the incoming message for the target component should be placed.
- The send destination is where the reply will be sent, unless the incoming message has overridden this using the replyTo header property.

A message listener is deployed to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply. The destination specified in the replyTo field of the incoming message overrides the destination specified in the send.

Figure 45 on page 125 illustrates how the external requestor is linked to the export.

Figure 45. JMS export binding resources

JMS integration and resource adapters:

The Java Message Service (JMS) provides integration through an available JMS JCA 1.5-based resource adapter. Complete support for JMS integration is provided for the Service Integration Bus (SIB) JMS resource adapter.

Use a JMS provider for JCA 1.5 resource adapter when you want to integrate with an external JCA 1.5-compliant JMS system. External services compliant with JCA 1.5 can receive messages and send messages to integrate with your service component architecture (SCA) components using the SIB JMS resource adapter.

The use of other provider-specific JCA 1.5 resource adapters is not supported.

JMS modules cannot be deployed to a Java SE environment. Such modules are only deployable to a Java EE environment.

Key features of JMS bindings:

Key features of JMS import and export bindings include headers and created Java EE resources.

Special headers

Special header properties are used in JMS imports and exports to tell the target how to handle the message.

For example, TargetFunctionName maps from the native method to the operation method.

Java EE resources

A number of Java EE resources are created when JMS imports and exports are deployed to a Java EE environment.

`ConnectionFactory`
Used by clients to create a connection to the JMS provider.

`ActivationSpec`
Imports use this for receiving the response to a request; exports use it when configuring the message endpoints that represent message listeners in their interactions with the messaging system.

`Destinations`

- Send destination: on an import, this is where the request or outgoing message is sent; on an export, this is the destination where the response message will be sent, if not superseded by the JMSReplyTo header field in the incoming message.
- Receive destination: where the incoming message should be placed; with imports, this is a response; with exports, this is a request.
- Callback destination: SCA JMS system destination used to store correlation information. Do not read or write to this destination.

The installation task creates the ConnectionFactory and three destinations. It also creates the ActivationSpec to enable the runtime message listener to listen for replies on the receive destination. The properties of these resources are specified in the import or export file.

JMS headers:

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

JMS system header

The JMS system header is represented in the SMO by the JMSHeader element, which contains all the fields typically found in a JMS header. Although these can be modified in the mediation (or ContextService), some JMS system header fields set in the SMO will not be propagated in the outbound JMS message as they are overridden by system or static values.

The key fields in the JMS system header that can be updated in a mediation (or ContextService) are:

- **JMSType** and **JMSCorrelationID** – values of the specific predefined message header properties
- **JMSDeliveryMode** – values for delivery mode (persistent or nonpersistent; default is persistent)

- **JMSPriority** – priority value (0 to 9; default is JMS_Default_Priority)

JMS properties

JMS properties are represented in the SMO as entries in the Properties list. The properties can be added, updated, or deleted in a mediation or by using the ContextService API.

Properties can also be set statically in the JMS binding. Properties that are set statically override settings (with the same name) that are set dynamically.

User properties propagated from other bindings (for example, an HTTP binding) will be output in the JMS binding as JMS properties.

Header propagation settings

The propagation of the JMS system header and properties either from the inbound JMS message to downstream components or from upstream components to the outbound JMS message can be controlled by the Propagate Protocol Header flag on the binding.

When Propagate Protocol Header is set, header information is allowed to flow to the message or to the target component, as described in the following list:

- JMS export request

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.
- JMS export response

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS export binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS export binding.
- JMS import request

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS import binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS import binding.
- JMS import response

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.

JMS temporary dynamic response destination correlation scheme:

The temporary dynamic response destination correlation scheme causes a unique dynamic queue or topic to be created for each request sent.

The static response destination specified in the import is used to derive the nature of the temporary dynamic destination queue or topic. This is set in the **ReplyTo** field of the request, and the JMS import listens for responses on that destination. When the response is received it is requeued to the static response destination for asynchronous processing. The **CorrelationID** field of the response is not used, and does not need to be set.

Transactional issues

When a temporary dynamic destination is being used, the response must be consumed in the same thread as the sent response. The request must be sent outside the global transaction, and must be committed before it is received by the backend service, and a response returned.

Persistence

Temporary dynamic queues are short-lived entities and do not guarantee the same level of persistence associated with a static queue or topic. A temporary dynamic queue or topic will not survive a server restart and neither will messages. After the message has been requeued to the static response destination it retains the persistence defined in the message.

Timeout

The import waits to receive the response on the temporary dynamic response destination for a fixed amount of time. This time interval will be taken from the SCA Response Expiration time qualifier, if it is set, otherwise the time defaults to 60 seconds. If the wait time is exceeded the import throws a `ServiceTimeoutRuntimeException`.

External clients:

The server can send messages to, or receive messages from, external clients using JMS bindings.

An external client (such as a Web portal or an enterprise information system) can send a message to an SCA module in the server, or it can be invoked by a component from within the server.

The JMS export components deploy message listeners to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications with the export binding.

JMS imports bind to, and can deliver messages to, external clients. This message might or might not demand a response from the external client.

Working with external clients:

An external client (that is, outside the server) might need to interact with an application installed in the server.

About this task

Consider a very simple scenario in which an external client wants to interact with an application on the server. The figure depicts a typical simple scenario.

Figure 46. Simple use-case scenario: external client interacts with server application

The SCA application includes an export with a JMS binding; this makes the application available to external clients.

When you have an external client in a Java virtual machine (JVM) separate from your server, there are several steps you must take in order to make a connection and interact with a JMS export. The client obtains an InitialContext with the correct values and then looks up the resources through JNDI. The client then uses the JMS 1.1 specification client to access the destinations and the send and receive messages on the destinations.

The default JNDI names of the resources created automatically by the runtime are listed in the configuration topic of this section. However, if you have pre-created resources, use those JNDI names.

Procedure

1. Configure JMS destinations and the connection factory to send the message.
2. Make sure that the JNDI context, the port for the SIB resource adapter, and the messaging bootstrapping port are correct.

   The server uses some default ports, but if there are more servers installed on that system, alternate ports are created at installation time to avoid conflicts with other server instances. You can use the administrative console to determine which ports your server is employing . Go to **Servers** > **Application Servers** > ***your_server_ name*** > **Configuration** and click **Ports** under **Communication**. You can then edit the port being used.
3. The client obtains an initial context with the correct values and then looks up the resources through JNDI.
4. Using JMS 1.1 specifications, the client accesses the destinations and the send and receive messages on the destinations.

Troubleshooting JMS bindings:

You can diagnose and fix problems with JMS bindings.

Implementation exceptions

In response to various error conditions, the JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

  For example, an import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

Primary failure conditions

The primary failure conditions of JMS bindings are determined by transactional semantics, by JMS provider configuration, or by reference to existing behavior in other components. The primary failure conditions include:

- Failure to connect to the JMS provider or destination.

  A failure to connect to the JMS provider to receive messages will result in the message listener failing to start. This condition will be logged in the WebSphere Application Server log. Persistent messages will remain on the destination until they are successfully retrieved (or expired).

  A failure to connect to the JMS provider to send outbound messages will cause rollback of the transaction controlling the send.
- Failure to parse an inbound message or to construct an outbound message.

  A failure in the data binding or data handler causes rollback of the transaction controlling the work.
- Failure to send the outbound message.

  A failure to send a message causes rollback of the relevant transaction.
- Multiple or unexpected late response messages.

  The import expects only one response message for each request message. Also the valid time period in which a response can be received is determined by the SCA Response Expiration qualifier on the request. When a response arrives or the expiration time is exceeded, the correlation record is deleted. If response messages arrive unexpectedly or arrive late, a Service Runtime Exception is thrown.
- Service timeout runtime exception caused by late response when using the temporary dynamic response destination correlation scheme.

  The JMS import will timeout after a period of time determined by the SCA response expiration qualifier, or if this is not set it will default to 60 seconds.

JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the underlying SIB destination of the JMS destination has a maximum failed deliveries value greater than 1. Setting this value to 2 or more enables interaction with the failed event manager during SCA invocations for the JMS bindings.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a `DataBindingException`.

When any runtime exception, including a `DataBindingException`, is thrown:

- If the mediation flow is configured to be transactional, the JMS message , by default, is stored in the Failed Event Manager for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the Failed Event Manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a `DataHandlerException` is reported to you as a `DataBindingException`.

Generic JMS bindings

The Generic JMS binding provides connectivity to third-party JMS 1.1 compliant providers. The operation of the Generic JMS bindings is similar to that of JMS bindings.

The service provided through a JMS binding allows a Service Component Architecture (SCA) module to make calls or receive messages from external systems. The system can be an external JMS system.

The Generic JMS binding provides integration with non-JCA 1.5-compliant JMS providers that support JMS 1.1 and implement the optional JMS Application Server Facility. The Generic JMS binding supports those JMS providers (including Oracle AQ, TIBCO, SonicMQ, WebMethods, BEA WebLogic, and WebSphere MQ) that do not support JCA 1.5 but do support the Application Server Facility of the JMS 1.1 specification. The WebSphere embedded JMS provider (SIBJMS), which is a JCA 1.5 JMS provider, is not supported by this binding; when using that provider, use the "JMS bindings" on page 122.

Use this Generic binding when integrating with a non-JCA 1.5-compliant JMS-based system within an SCA environment. The target external applications can then receive messages and send messages to integrate with an SCA component.

Generic JMS bindings overview:

Generic JMS bindings are non-JCA JMS bindings that provide connectivity between the Service Component Architecture (SCA) environment and JMS systems that are compliant with JMS 1.1 and that implement the optional JMS Application Server Facility.

Generic JMS bindings

The major aspects of Generic JMS import and export bindings are:

- Listener port: enables non-JCA-based JMS providers to receive messages and dispatch them to a Message Driven Bean (MDB)
- Connections: encapsulate a virtual connection between a client and a provider application
- Destinations: used by a client to specify the target of messages it produces or the source of messages it consumes
- Authentication data: used to secure access to the binding

Generic JMS import bindings

Generic JMS import bindings allow components within your SCA module to communicate with services provided by external non-JCA 1.5-compliant JMS providers.

The connection part of a JMS import is a connection factory. A connection factory, the object a client uses to create a connection to a provider, encapsulates a set of connection configuration parameters defined by an administrator. Each connection factory is an instance of the ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory interface.

Interaction with external JMS systems includes the use of destinations for sending requests and receiving replies.

Two types of usage scenarios for the Generic JMS import binding are supported, depending on the type of operation being invoked:

- One-way: The Generic JMS import puts a message on the send destination configured in the import binding Nothing is sent to the replyTo field of the JMS header.
- Two-way (request-response): The Generic JMS import puts a message on the send destination and then persists the reply it receives from the SCA component.

  The receive destination is set in the replyTo header property of the outbound message. A message driven bean (MDB) is deployed to listen on the receive destination, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified. Static properties can be set from the Generic JMS import method binding Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that Generic JMS is an asynchronous binding. If a calling component invokes a Generic JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 47 on page 133 illustrates how the import is linked to the external service.

Figure 47. Generic JMS import binding resources

Generic JMS export bindings

Generic JMS export bindings provide the means for SCA modules to provide services to external JMS applications.

The connection part of a JMS export is composed of a ConnectionFactory and a ListenerPort.

A Generic JMS export has send and receive destinations.

- The receive destination is where the incoming message for the target component should be placed.
- The send destination is where the reply will be sent, unless the incoming message has overridden this using the replyTo header property.

An MDB is deployed to listen to requests incoming to the receive destination specified in the export binding.

- The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply.
- The destination specified in the replyTo field of the incoming message overrides the destination specified in the send field.
- For request/response scenarios, the import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response to copy the request message ID to the correlation ID field of the response message (default), or the response can copy the request correlation ID to the correlation ID field of the response message.

Figure 48 illustrates how the external requestor is linked to the export.

Figure 48. Generic JMS export binding resources

Key features of Generic JMS bindings:

The features of the Generic JMS import and export binding are consistent with those of the WebSphere embedded JMS and MQ JMS import bindings. Key features include header definitions and access to existing Java EE resources. However, because of its generic nature, there are no JMS provider-specific connectivity options, and this binding has limited capability to generate resources at deployment and installation.

Generic imports

Like the MQ JMS import application, the Generic JMS implementation is asynchronous and supports three invocations: one-way, two-way (also known as request-response), and callback.

When the JMS import is deployed, a message driven bean (MDB) provided by the runtime environment is deployed. The MDB listens for replies to the request message. The MDB is associated with (listens on) the destination sent with the request in the replyTo header field of the JMS message.

Generic exports

Generic JMS export bindings differ from EIS export bindings in their handling of the return of the result. A Generic JMS export explicitly sends the response to the replyTo destination specified on the incoming message. If none is specified, the send destination is used.

When the Generic JMS export is deployed, a message driven bean (a different MDB than the one used for Generic JMS imports) is deployed. It listens for the incoming requests on the receive destination and then dispatches the requests to be processed by the SCA runtime.

Special headers

Special header properties are used in Generic JMS imports and exports to tell the target binding how to handle the message.

For example, the TargetFunctionName property is used by the default function selector to identify the name of the operation in the export interface that is being invoked.

Note: The import binding can be configured to set the TargetFunctionName header to the operation name for each operation.

Java EE resources

A number of Java EE resources are created when a JMS binding is deployed to a Java EE environment.

- Listener port for listening on the receive (response) destination (two-way only) for imports and on the receive (request) destination for exports
- Generic JMS connection factory for the outboundConnection (import) and inboundConnection (export)
- Generic JMS destination for the send (import) and receive (export; two-way only) destinations
- Generic JMS connection factory for the responseConnection (two-way only and optional; otherwise, outboundConnection is used for imports, and inboundConnection is used for exports)
- Generic JMS destination for the receive (import) and send (export) destination (two-way only)
- Default messaging provider callback JMS destination used to access the SIB callback queue destination (two-way only)
- Default messaging provider callback JMS connection factory used to access the callback JMS destination (two-way only)
- SIB callback queue destination used to store information about the request message for use during response processing (two-way only)

The installation task creates the ConnectionFactory, the three destinations, and the ActivationSpec from the information in the import and export files.

Generic JMS headers:

Generic JMS headers are Service Data Objects (SDO) that contain all the properties of the Generic JMS message properties. These properties can be from the inbound message or they can be the properties that will be applied to the outbound message.

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

The following properties are set statically on the methodBinding:

- JMSType
- JMSCorrelationID
- JMSDeliveryMode
- JMSPriority

The Generic JMS binding also supports dynamic modification of JMS headers and properties in the same manner as the JMS and MQ JMS bindings.

Some Generic JMS providers place restrictions on which properties can be set by the application and in what combinations. You must consult your third-party product documentation for more information. However, an additional property has been added to the methodBinding, ignoreInvalidOutboundJMSProperties, which allows any exceptions to be propagated.

The Generic JMS headers and message properties are used only when the base service component architecture SCDL binding switch is turned on. When the switch is turned on, context information is propagated. By default, this switch is on. To prevent context information propagation, change the value to `false`.

When context propagation is enabled, header information is allowed to flow to the message or to the target component. To turn on and off context propagation, specify `true` or `false` for the contextPropagationEnabled attribute of the import and export bindings. For example:

```
<esbBinding xsi:type="eis:JMSImportBinding" contextProgagationEnabled="true">
```

The default is `true`.

Troubleshooting Generic JMS bindings:

You can diagnose and fix problems with Generic JMS binding.

Implementation exceptions

In response to various error conditions, the Generic JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

Troubleshooting Generic JMS message expiry

A request message by the JMS provider is subject to expiration.

Request expiry refers to the expiration of a request message by the JMS provider when the JMSExpiration time on the request message is reached. As with other JMS bindings, the Generic JMS binding handles the request expiry by setting expiration on the callback message placed by the import to be the same as for the outgoing request. Notification of the expiration of the callback message will indicate that the request message has expired and the client should be notified by means of a business exception.

If the callback destination is moved to the third-party provider, however, this type of request expiry is not supported.

Response expiry refers to the expiration of a response message by the JMS provider when the JMSExpiration time on the response message is reached.

Response expiry for the generic JMS binding is not supported, because the exact expiry behavior of a third-party JMS provider is not defined. You can, however, check that the response is not expired if and when it is received.

For outbound request messages, the JMSExpiration value will be calculated from the time waited and from the requestExpiration values carried in the asyncHeader, if set.

Troubleshooting Generic JMS connection factory errors

When you define certain types of connection factories in your Generic JMS provider, you might receive an error message when you try to start an application. You can modify the external connection factory to avoid this problem.

When launching an application, you might receive the following error message:

```
MDB Listener Port JMSConnectionFactory type does not match
JMSDestination type
```

This problem can arise when you are defining external connection factories. Specifically, the exception can be thrown when you create a JMS 1.0.2 Topic Connection Factory, instead of a JMS 1.1 (unified) Connection Factory (that is, one that is able to support both point-to-point and publish/subscribe communication).

To resolve this issue, take the following steps:

1. Access the Generic JMS provider that you are using.
2. Replace the JMS 1.0.2 Topic Connection Factory that you defined with a JMS 1.1 (unified) Connection Factory.

When you launch the application with the newly defined JMS 1.1 Connection Factory, you should no longer receive an error message.

Generic JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a generic JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the value of the maximum retries property on the underlying listener port is equal to or greater than 1. Setting this value to 1 or more enables interaction with the failed event manager during SCA invocations for the generic JMS bindings.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a `DataBindingException`.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a `DataHandlerException` is reported to you as a `DataBindingException`.

When any runtime exception, including a `DataBindingException` exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a `DataHandlerException` is reported to you as a `DataBindingException`.

WebSphere MQ JMS bindings

The WebSphere MQ JMS binding provides integration with external applications that use a WebSphere MQ JMS-based provider.

Use the WebSphere MQ JMS export and import bindings to integrate directly with external JMS or MQ JMS systems from your server environment. This eliminates the need to use MQ Link or Client Link features of the Service Integration Bus.

When a component interacts with a WebSphere MQ JMS-based service by way of an import, the WebSphere MQ JMS import binding utilizes a destination to which data will be sent and a destination where the reply can be received. Conversion of the data to and from a JMS message is accomplished through the JMS data handler or data binding edge component.

When an SCA module provides a service to WebSphere MQ JMS clients, the WebSphere MQ JMS export binding utilizes a destination where the request can be received and the response can be sent. The conversion of the data to and from a JMS message is done through the JMS data handler or data binding.

The function selector provides a mapping to the operation on the target component to be invoked.

WebSphere MQ JMS bindings overview:

The WebSphere MQ JMS binding provides integration with external applications that use the WebSphere MQ JMS provider.

WebSphere MQ administrative tasks

The WebSphere MQ system administrator is expected to create the underlying WebSphere MQ Queue Manager, which the WebSphere MQ JMS bindings will use, before running an application containing these bindings.

WebSphere MQ JMS import bindings

The WebSphere MQ JMS import allows components within your SCA module to communicate with services provided by WebSphere MQ JMS-based providers. You must be using a supported version of WebSphere MQ. Detailed hardware and software requirements can be found on the IBM support pages.

Two types of usage scenarios for WebSphere MQ JMS import bindings are supported, depending on the type of operation being invoked:

- One-way: The WebSphere MQ JMS import puts a message on the send destination configured in the import binding. Nothing is sent to the replyTo field of the JMS header.
- Two-way (request-response): The WebSphere MQ JMS import puts a message on the send destination.

  The receive destination is set in the replyTo header field. A message-driven bean (MDB) is deployed to listen on the receive destination, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified.Static properties can be set from the JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that WebSphere MQ JMS is an asynchronous binding. If a calling component invokes a WebSphere MQ JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 49 on page 140 illustrates how the import is linked to the external service.

Figure 49. WebSphere MQ JMS import binding resources

WebSphere MQ JMS export bindings

The WebSphere MQ JMS export binding provides the means for SCA modules to provide services to external JMS applications on the WebSphere MQ-based JMS provider.

An MDB is deployed to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply. The destination specified in the replyTo field of the response message overrides the destination specified in the send field.

Figure 50 on page 141 illustrates how the external requestor is linked to the export.

Figure 50. WebSphere MQ JMS export binding resources

Note: Figure 49 on page 140 and Figure 50 illustrate how an application from a previous version of WebSphere ESB is linked to an external service. For applications developed for WebSphere ESB Version 7.0, the Activation specification is used instead of the Listener Port and Connection Factory.

Key features of WebSphere MQ JMS bindings:

Key features of WebSphere MQ JMS bindings include headers, Java EE artifacts, and created Java EE resources.

Headers

A JMS message header contains a number of predefined fields containing values used by both clients and providers to identify and to route messages. You can use binding properties to configure these headers with fixed values, or the headers can be specified dynamically at runtime.

`JMSCorrelationID`
Links to a related message. Typically, this field is set to the message identifier string of the message that is being replied to.

`TargetFunctionName`
This header is used by one of the supplied function selectors to identify the operation being invoked. Setting the TargetFunctionName JMS header property in messages sent to a JMS export allows this function selector to be used. The property can be set directly in JMS client applications or when connecting an import with a JMS binding to such an export. In this case, the JMS import binding should be configured to set the TargetFunctionName header for each operation in the interface to the name of the operation.

Correlation schemes

The WebSphere MQ JMS bindings provide various correlation schemes that are used to determine how to correlate request messages with response messages.

RequestMsgIDToCorrelID
The JMSMessageID is copied to the JMSCorrelationID field. This is the default setting.

RequestCorrelIDToCorrelID
The JMSCorrelationID is copied to the JMSCorrelationID field.

Java EE resources

A number of Java EE resources are created when an MQ JMS import is deployed to a Java EE environment.

Parameters

MQ Connection Factory
Used by clients to create a connection to the MQ JMS provider.

Response Connection Factory
Used by the SCA MQ JMS runtime when the send destination is on a different Queue Manager than the receive destination.

Activation specification
An MQ JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

Destinations

- Send destination:
  - Imports: Where the request or outgoing message is sent.
  - Exports: Where the response message will be sent if it is not superseded by the JMSReplyTo header field of the incoming message.
- Receive destination:
  - Imports: Where the response or incoming message should be placed.
  - Exports: Where the incoming or request message should be placed.

JMS headers:

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

JMS system header

The JMS system header is represented in the SMO by the JMSHeader element, which contains all the fields typically found in a JMS header. Although these can be modified in the mediation (or ContextService), some JMS system header fields set in the SMO will not be propagated in the outbound JMS message as they are overridden by system or static values.

The key fields in the JMS system header that can be updated in a mediation (or ContextService) are:

- **JMSType** and **JMSCorrelationID** – values of the specific predefined message header properties
- **JMSDeliveryMode** – values for delivery mode (persistent or nonpersistent; default is persistent)
- **JMSPriority** – priority value (0 to 9; default is JMS_Default_Priority)

JMS properties

JMS properties are represented in the SMO as entries in the Properties list. The properties can be added, updated, or deleted in a mediation or by using the ContextService API.

Properties can also be set statically in the JMS binding. Properties that are set statically override settings (with the same name) that are set dynamically.

User properties propagated from other bindings (for example, an HTTP binding) will be output in the JMS binding as JMS properties.

Header propagation settings

The propagation of the JMS system header and properties either from the inbound JMS message to downstream components or from upstream components to the outbound JMS message can be controlled by the Propagate Protocol Header flag on the binding.

When Propagate Protocol Header is set, header information is allowed to flow to the message or to the target component, as described in the following list:

- JMS export request

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.
- JMS export response

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS export binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS export binding.
- JMS import request

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS import binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS import binding.
- JMS import response

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.

External clients:

The server can send messages to, or receive messages from, external clients using WebSphere MQ JMS bindings.

An external client (such as a Web portal or an enterprise information system) can send a message to an SCA component in the application by way of an export or it can be invoked by an SCA component in the application by way of an import.

The WebSphere MQ JMS export binding deploys message driven beans (MDBs) to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications via the export binding.

WebSphere MQ JMS imports bind to, and can deliver message to, external clients. This message might or might not demand a response from the external client.

More information on how to interact with external clients using WebSphere MQ can be found at the WebSphere MQ information center.

Troubleshooting WebSphere MQ JMS bindings:

You can diagnose and fix problems with WebSphere MQ JMS bindings.

Implementation exceptions

In response to various error conditions, the MQ JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

  For example, an import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

WebSphere MQ JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a WebSphere MQ JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the value of the maximum retries property on the underlying listener port is equal to or greater than 1. Setting this value to 1 or more enables interaction with the failed event manager during SCA invocations for the MQ JMS bindings.

Misusage scenarios: comparison with WebSphere MQ bindings

The WebSphere MQ JMS binding is designed to interoperate with JMS applications deployed against WebSphere MQ, which exposes messages according to the JMS message model. The WebSphere MQ import and export, however, are principally designed to interoperate with native WebSphere MQ applications and expose the full content of the WebSphere MQ message body to mediations.

The following scenarios should be built using the WebSphere MQ JMS binding, not the WebSphere MQ binding:

- Invoking a JMS message-driven bean (MDB) from an SCA module, where the MDB is deployed against the WebSphere MQ JMS provider. Use a WebSphere MQ JMS import.
- Allowing the SCA module to be called from a Java EE component servlet or EJB by way of JMS. Use a WebSphere MQ JMS export.

- Mediating the contents of a JMS MapMessage, in transit across WebSphere MQ. Use a WebSphere MQ JMS export and import in conjunction with the appropriate data handler or data binding.

There are situations in which the WebSphere MQ binding and WebSphere MQ JMS binding might be expected to interoperate. In particular, when you are bridging between Java EE and non-Java EE WebSphere MQ applications, use a WebSphere MQ export and WebSphere MQ JMS import (or vice versa) in conjunction with appropriate data bindings or mediation modules (or both).

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a `DataBindingException`.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a `DataHandlerException` is reported to you as a `DataBindingException`.

When any runtime exception, including a `DataBindingException` exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a `DataHandlerException` is reported to you as a `DataBindingException`.

WebSphere MQ bindings

The WebSphere MQ binding provides Service Component Architecture (SCA) connectivity with WebSphere MQ applications.

Use the WebSphere MQ export and import bindings to integrate directly with a WebSphere MQ-based system from your server environment. This eliminates the need to use MQ Link or Client Link features of the Service Integration Bus.

When a component interacts with a WebSphere MQ service by way of an import, the WebSphere MQ import binding uses a queue to which data will be sent and a queue where the reply can be received.

When an SCA module provides a service to WebSphere MQ clients, the WebSphere MQ export binding uses a queue where the request can be received and the response can be sent. The function selector provides a mapping to the operation on the target component to be invoked.

Conversion of the payload data to and from an MQ message is done through the MQ body data handler or data binding. Conversion of the header data to and from an MQ message is done through the MQ header data binding.

For information about the WebSphere MQ versions supported, see the system requirements web page: WebSphere ESB system requirements

WebSphere MQ bindings overview:

The WebSphere MQ binding provides integration with native MQ-based applications.

WebSphere MQ administrative tasks

The WebSphere MQ system administrator is expected to create the underlying WebSphere MQ Queue Manager, which the WebSphere MQ bindings will use, before running an application containing these bindings.

WebSphere administrative tasks

You must set the **Native library path** property of the MQ resource adapter in Websphere to the WebSphere MQ version supported by the server, and restart the server. This ensures that the libraries of a supported version of WebSphere MQ are being used. Detailed hardware and software requirements can be found on the IBM support pages

WebSphere MQ import bindings

The WebSphere MQ import binding allows components within your SCA module to communicate with services provided by external WebSphere MQ-based applications. You must be using a supported version of WebSphere MQ. Detailed hardware and software requirements can be found on the IBM support pages.

Interaction with external WebSphere MQ systems includes the use of queues for sending requests and receiving replies.

Two types of usage scenarios for the WebSphere MQ import binding are supported, depending on the type of operation being invoked:

- One-way: The WebSphere MQ import puts a message on the queue configured in the **Send destination queue** field of the import binding. Nothing is sent to the replyTo field of the MQMD header.
- Two-way (request-response): The WebSphere MQ import puts a message on the queue configured in the **Send destination queue** field

  The receive queue is set in the replyTo MQMD header field. A message driven bean (MDB) is deployed to listen on the receive queue, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

It is important to note that WebSphere MQ is an asynchronous binding. If a calling component invokes a WebSphere MQ import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the WebSphere MQ service.

Figure 51 illustrates how the import is linked to the external service.

Figure 51. WebSphere MQ import binding resources

WebSphere MQ export bindings

The WebSphere MQ export binding provides the means for SCA modules to provide services to external WebSphere MQ-based applications.

An MDB is deployed to listen to requests incoming to the **Receive destination queue** specified in the export binding. The queue specified in the **Send destination queue** field is used to send the reply to the inbound request if the invoked component provides a reply. The queue specified in the replyTo field of the response message overrides the queue specified in the **Send destination queue** field.

Figure 52 on page 148 illustrates how the external requestor is linked to the export.

Figure 52. WebSphere MQ export binding resources

Note: Figure 51 on page 147 and Figure 52 illustrate how an application from a previous version of WebSphere ESB is linked to an external service. For applications developed for WebSphere ESB Version 7.0, the Activation specification is used instead of the Listener Port and Connection Factory.

Key features of a WebSphere MQ binding:

Key features of a WebSphere MQ binding include headers, Java EE artifacts, and created Java EE resources.

Correlation schemes

A WebSphere MQ request/reply application can use one of a number of techniques to correlate response messages with requests, built around the MQMD MessageID and CorrelID fields. In the vast majority of cases, the requestor lets the queue manager select a MessageID and expects the responding application to copy this into the CorrelID of the response. In most cases, the requestor and responding application implicitly know which correlation technique is in use. Occasionally the responding application will honor various flags in the Report field of the request that describe how to handle these fields.

Export bindings for WebSphere MQ messages can be configured with the following options:

Response MsgId options:

New MsgID
Allows the queue manager to select a unique MsgId for the response (default).

Copy from Request MsgID
Copies the MsgId field from the MsgId field in the request.

Copy from SCA message
Sets the MsgId to be that carried in WebSphere MQ headers in the SCA response message, or lets the queue manager define a new Id if the value does not exist.

As Report Options
Inspects the Report field of the MQMD in the request for a hint as to how to handle the MsgId. The MQRO_NEW_MSG_ID and MQRO_PASS_MSG_ID options are supported and behave like New MsgId and Copy from Request MsgID, respectively

Response CorrelId options:

Copy from Request MsgID
Copies the CorrelId field from the MsgId field in the request (default).

Copy from Request CorrelID
Copies the CorrelId field from the CorrelId field in the request.

Copy from SCA message
Sets the CorrelId to be carried in WebSphere MQ headers in the SCA response message or leaves it blank if the value does not exist.

As Report Options
Inspects the Report field of the MQMD in the request for a hint as to how to handle the CorrelId. The MQRO_COPY_MSG_ID_TO_CORREL_ID and MQRO_PASS_CORREL_ID options are supported and behave like Copy from Request MsgID and Copy from Request CorrelID, respectively

Import bindings for WebSphere MQ messages can be configured with the following options:

Request MsgId options:

New MsgID
Allows the queue manager to select a unique MsgId for the request (default)

Copy from SCA message
Sets the MsgId to be carried in WebSphere MQ headers in the SCA request message or lets the queue manager define a new Id if the value does not exist.

Response correlation options:

Response has CorrelID copied from MsgId
Expects the response message to have a CorrelId field set, per the MsgId of the request (default).

Response has MsgID copied from MsgId
Expects the response message to have a MsgId field set, per the MsgId of the request.

Response has CorrelID copied from CorrelId
Expects the response message to have a CorrelId field set, per the CorrelId of the request.

Java EE resources

A number of Java EE resources are created when a WebSphere MQ binding is deployed to a Java EE environment.

Parameters

MQ Connection Factory
Used by clients to create a connection to the WebSphere MQ provider.

Response Connection Factory
Used by the SCA MQ runtime when the send destination is on a different Queue Manager than the receive destination.

Activation specification
An MQ JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

Destinations

- Send destination: where the request or outgoing message is sent (import); where the response message will be sent (export), if not superseded by the MQMD ReplyTo header field in the incoming message.
- Receive destination: where the response/request or incoming message should be placed.

WebSphere MQ headers:

WebSphere MQ headers incorporate certain conventions for conversion to the service component architecture (SCA) messages.

WebSphere MQ messages consist of a system header (the MQMD), zero or more other MQ headers (system or custom), and a message body. If multiple message headers exist in the message, the order of the headers is significant.

Each header contains information describing the structure of the following header. The MQMD describes the first header.

How MQ headers are parsed

An MQ Header data binding is used to parse MQ headers. The following headers are supported automatically:

- MQRFH
- MQRFH2
- MQCIH
- MQIIH

Headers that start with `MQH` are handled differently. Specific fields of the header are not parsed; they remain as unparsed bytes.

For other MQ headers, you can write custom MQ header data bindings to parse those headers.

How MQ headers are accessed

MQ headers can be accessed in the product in one of two ways:

- Through the service message object (SMO) in a mediation
- Through the ContextService API

MQ headers are represented internally with the SMO MQHeader element. MQHeader is a container of header data that extends MQControl but contains a value element of anyType. It contains the MQMD, MQControl (MQ message body control information), and a list of other MQ headers.

- MQMD represents the contents of the WebSphere MQ message description, except for information determining the structure and encoding of the body.
- MQControl contains information determining the structure and encoding of a message body.
- MQHeaders contain a list of MQHeader objects.

The MQ header chain is unwound so that, inside the SMO, each MQ header carries its own control information (CCSID, Encoding, and Format). Headers can be added or deleted easily, without altering other header data.

Setting fields in the MQMD

You can update the MQMD using the Context API or through the service message object (SMO) in a mediation. The following fields are automatically propagated to the outbound MQ message:

- Encoding
- CodedCharacterSet
- Format
- Report
- Expiry
- Feedback
- Priority
- Persistence
- CorrelId
- MsgFlags

Configure the MQ binding on an Import or Export to propagate the following properties to the outbound MQ message:

`MsgID`
: Set **Request Message ID** to `copy from SCA message`.

`MsgType`
: Clear the **Set message type to MQMT_DATAGRAM or MQMT_REQUEST for request-response operation** check box.

`ReplyToQ`
: Clear the **Override reply to queue of request message** check box.

`ReplyToQMgr`
: Clear the **Override reply to queue of request message** check box.

From version 7.0 onwards, context fields can be overridden using a custom property on the JNDI destination definition. Set the custom property MDCTX with value `SET_IDENTITY_CONTEXT` on the send destination to propagate the following fields to the outbound MQ message:

- UserIdentifier
- AppIdentityData

Set the custom property MDCTX with value `SET_ALL_CONTEXT` on the send destination to propagate the following properties to the outbound MQ message:

- UserIdentifier
- AppIdentityData
- PutApplType
- PutApplName
- ApplOriginData

Some fields are not propagated to the outbound MQ message. The following fields are overridden during the send of the message:

- BackoutCount
- AccountingToken
- PutDate
- PutTime
- Offset
- OriginalLength

Adding MQCIH statically in a WebSphere MQ binding:

WebSphere ESB supports adding MQCIH header information statically without using a mediation module.

There are various ways to add MQCIH header information to a message (for example, by using the Header Setter mediation primitive). It might be useful to add this header information statically, without the use of an additional mediation module. Static header information, including the CICS® program name, the transaction ID, and other data format header details, can be defined and added as part of the WebSphere MQ binding.

WebSphere MQ, the MQ CICS Bridge, and CICS must be configured for MQCIH header information to be added statically.

You can use Integration Designer to configure the WebSphere MQ import with the static values that are required for the MQCIH header information.

When a message arrives and is processed by the WebSphere MQ import, a check is made to see if MQCIH header information is already present in the message. If the MQCIH is present, the static values defined in the WebSphere MQ import are used to override the corresponding dynamic values in the message. If the MQCIH is not present, one is created in the message and the static values defined in the WebSphere MQ import are added.

The static values defined in the WebSphere MQ import are specific to a method. You can specify different static MQCIH values for different methods within the same WebSphere MQ import.

This facility is not used to provide default values if the MQCIH does not contain specific header information because a static value defined in the WebSphere MQ import will override a corresponding value provided in the incoming message.

External clients:

WebSphere ESB can send messages to, or receive messages from, external clients using WebSphere MQ bindings.

An external client (for example, a Web portal or an enterprise information system) can send a message to an SCA component in the application by way of an export or it can be invoked by an SCA component in the application by way of an import.

The WebSphere MQ export binding deploys message driven beans (MDBs) to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications by way of the export binding.

WebSphere MQ imports bind to, and can deliver message to, external clients. This message might or might not demand a response from the external client.

More information on how to interact with external clients using WebSphere MQ can be found at the WebSphere MQ information center.

Troubleshooting WebSphere MQ bindings:

You can diagnose and fix faults and failure conditions that occur with WebSphere MQ bindings.

Primary failure conditions

The primary failure conditions of WebSphere MQ bindings are determined by transactional semantics, by WebSphere MQ configuration, or by reference to existing behavior in other components. The primary failure conditions include:

- Failure to connect to the WebSphere MQ queue manager or queue.

  A failure to connect to WebSphere MQ to receive messages will result in the MDB Listener Port failing to start. This condition will be logged in the WebSphere Application Server log. Persistent messages will remain on the WebSphere MQ queue until they are successfully retrieved (or expired by WebSphere MQ).

  A failure to connect to WebSphere MQ to send outbound messages will cause rollback of the transaction controlling the send.
- Failure to parse an inbound message or to construct an outbound message.

  A failure in the data binding causes rollback of the transaction controlling the work.
- Failure to send the outbound message.

  A failure to send a message causes rollback of the relevant transaction.
- Multiple or unexpected response messages.

  The import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

Misusage scenarios: comparison with WebSphere MQ JMS bindings

The WebSphere MQ import and export are principally designed to interoperate with native WebSphere MQ applications and expose the full content of the WebSphere MQ message body to mediations. The WebSphere MQ JMS binding, however, is designed to interoperate with JMS applications deployed against WebSphere MQ, which exposes messages according to the JMS message model.

The following scenarios should be built using the WebSphere MQ JMS binding, not the WebSphere MQ binding:

- Invoking a JMS message-driven bean (MDB) from an SCA module, where the MDB is deployed against the WebSphere MQ JMS provider. Use a WebSphere MQ JMS import.
- Allowing the SCA module to be called from a Java EE component servlet or EJB by way of JMS. Use a WebSphere MQ JMS export.
- Mediating the contents of a JMS MapMessage, in transit across WebSphere MQ. Use a WebSphere MQ JMS export and import in conjunction with the appropriate data binding.

There are situations in which the WebSphere MQ binding and WebSphere MQ JMS binding might be expected to interoperate. In particular, when you are bridging between Java EE and non-Java EE WebSphere MQ applications, use a WebSphere MQ export and WebSphere MQ JMS import (or vice versa) in conjunction with appropriate data bindings or mediation modules (or both).

Undelivered messages

If WebSphere MQ cannot deliver a message to its intended destination (because of configuration errors, for example), it sends the messages instead to a nominated dead-letter queue.

In doing so, it adds a dead-letter header to the start of the message body. This header contains the failure reasons, the original destination, and other information.

MQ-based SCA messages not appearing in the failed event manager

If SCA messages originated because of a WebSphere MQ interaction failure, you would expect to find these messages in the failed event manager. If these messages are not showing in the failed event manager, check that the underlying WebSphere MQ destination has a maximum failed deliveries value greater than 1. Setting this value to 2 or more allows interaction with the failed event manager during SCA invocations for the WebSphere MQ bindings.

MQ failed events are replayed to the wrong queue manager

When a predefined connection factory is to be used for outbound connections, the connection properties must match those defined in the activation specification used for inbound connections.

The predefined connection factory is used to create a connection when replaying a failed event and must therefore be configured to use the same queue manager from which the message was originally received.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a `DataBindingException`.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a `DataHandlerException` is reported to you as a `DataBindingException`.

When any runtime exception, including a `DataBindingException` exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a `DataHandlerException` is reported to you as a `DataBindingException`.

Limitations of bindings

The bindings have some limitations in their use that are listed here.

Limitations of the MQ binding:

The MQ binding has some limitations in its use that are listed here.

No publish-subscribe message distribution

The publish-subscribe method of distributing messages is not currently supported by the MQ binding though WMQ itself supports publish-subscribe. However, the MQ JMS binding does support this method of distribution.

Shared receive queues

Multiple WebSphere MQ export and import bindings expect that any messages present on their configured receive queue are intended for that export or import. Import and export bindings should be configured with the following considerations:

- Each MQ import must have a different receive queue because the MQ import binding assumes all messages on the receive queue are responses to requests

that it sent. If the receive queue is shared by more than one import, responses could be received by the wrong import and will fail to be correlated with the original request message.
- Each MQ export should have a different receive queue, because otherwise you cannot predict which export will get any particular request message.
- MQ imports and exports can point to the same send queue.

Limitations of the JMS, MQ JMS, and generic JMS bindings:

The JMS and MQ JMS bindings have some limitations.

Implications of generating default bindings

The limitations of using the JMS, MQ JMS, and generic JMS bindings are discussed in the following sections:
- Implications of generating default bindings
- Response correlation scheme
- Bidirectional support

When you generate a binding, several fields will be filled in for you as defaults, if you do not choose to enter the values yourself. For example, a connection factory name will be created for you. If you know that you will be putting your application on a server and accessing it remotely with a client, you should at binding creation time enter JNDI names rather than take the defaults since you will likely want to control these values through the administrative console at run time.

However, if you did accept the defaults and then find later that you cannot access your application from a remote client, you can use the administrative console to explicitly set the connection factory value. Locate the provider endpoints field in the connection factory settings and add a value such as <server_hostname>:7276 (if using the default port number).

Response correlation scheme

If you use the CorrelationId To CorrelationId response correlation scheme, used to correlate messages in a request-response operation, you must have a dynamic correlation ID in the message.

To create a dynamic correlation ID in a mediation module using the mediation flow editor, add an XSLT node before the import with the JMS binding. Open the XSLT mapping editor. The known service component architecture headers will be available in the target message. Drag and drop a field containing a unique ID in the source message onto the correlation ID in the JMS header in the target message.

Bidirectional support

Only ASCII characters are supported for Java Naming and Directory Interface (JNDI) names at runtime.

Shared receive queues

Multiple export and import bindings expect that any messages present on their configured receive queue are intended for that export or import. Import and export bindings should be configured with the following considerations:

- Each import binding must have a different receive queue because the import binding assumes all messages on the receive queue are responses to requests that it sent. If the receive queue is shared by more than one import, responses could be received by the wrong import and will fail to be correlated with the original request message.
- Each export should have a different receive queue, because otherwise you cannot predict which export will get any particular request message.
- Imports and exports can point to the same send queue.

Relationships

A relationship is an association between two or more data entities, typically business objects. Relationships can be used to transform data that is equivalent across business objects and other data but that is represented differently, or they can be used to draw associations across different objects found in different applications. They can be shared across applications, across solutions, and even across products.

The relationship service in WebSphere Enterprise Service Bus provides the infrastructure and operations for managing relationships. Because it enables you to deal with business objects regardless of where they reside, it can provide a unified holistic view across all applications in an enterprise, and serve as a building block for BPM solutions. Because relationships are extensible and manageable, they can be used in complex integration solutions.

What are relationships?

A relationship is an association between business objects. Each business object in a relationship is called a *participant* in the relationship. Each participant in the relationship is distinguished from other participants based on the function, or *role*, it serves in that relationship. A relationship contains a list of roles.

The relationship *definition* describes each role and specifies how the roles are related. It also describes the overall "shape" of the relationship. For example, this role can have only one participant, but this other role can have as many participants as necessary. You might define a *car-owner* relationship, for instance, where one owner might own multiple cars. For example, one instance could have the following participants for each of these roles:

- Car (Ferrari)
- Owner (John)

The relationship definition is a template for the relationship *instance*. The instance is the run-time instantiation of the relationship. In the *car-owner* example, an instance might describe any of the following associations:

- John owns Ferrari
- Sara owns Mazda
- Bob owns Ferrari

Using relationships frees you from the need to custom build relationship tracking persistence within your business logic. For certain scenarios, the relationship service does all the work for you. See the example described in the section on Identity relationships.

Scenarios

Here is a typical example of a situation in which an integration solution might use relationships. A large corporation buys multiple companies, or business units. Each business unit uses different software to monitor personnel and laptops. The company needs a way to monitor its employees and their laptops. It wants a solution that enables them to:

- View all the employees in the various business units as if they were in one database
- Have a single view of all their laptops
- Allow employees to log on to the system and buy a laptop
- Accommodate the different enterprise application systems in the various business units

To accomplish this, the company needs a way to ensure, for example, that John Smith and John A. Smith in different applications are seen as the same employee. For Example, they need a way to consolidate a single entity across multiple application spaces.

More complex relationship scenarios involve building BPEL processes that draw relationships across different objects found in multiple applications. With complex relationship scenarios, the business objects reside in the integration solution, and not in the applications. The relationship service provides a platform for managing relationships persistently. Before the relationship service, you would have to build your own object persistence service. Two examples of complex relationship scenarios are:

- You have a `car` business object with a VIN number in an SAP application, and you want to track the fact that this car is owned by someone else. However, the ownership relationship is with someone in a PeopleSoft application. In this pattern of relationships, you have two solutions and you need to build a cross-bridge between them.
- A large retail company wants to be able to monitor merchandise returned for cash back or credit. There are two different applications involved: an order management system (OMS) for purchases, and a returns management system (RMS) for returns. The business objects reside in more than one application, and you need a way to show the relationships that exist between them.

Common usage patterns

The most common relationship patterns are *equivalence* patterns. These are based on cross-referencing, or correlation. There are two types of relationships that fit this pattern: *non-identity* and *identity*.

- **Non-identity relationships** establish associations between business objects or other data on a one-to-many or many-to-many basis. For each relationship instance, there can be one or more instances of each participant. One type of non-identity relationship is a static lookup relationship. An example of this is a relationship in which `CA` in an SAP application is related to `California` in a Siebel application.
-

Identity relationships establish associations between business objects or other data on a one-to-one basis. For each relationship instance, there can be only one instance of each participant. Identity relationships capture cross-references between business objects that are semantically equivalent, but that are identified differently within different applications. Each participant in the relationship is associated with a business object that has a value (or combination of values) that uniquely identifies the object. Identity relationships typically transform the key attributes of business objects, such as ID numbers and product codes.

For example, if you have `car` business objects in SAP, PeopleSoft, and Siebel applications, and you want to build a solution that synchronizes them, you would normally need to introduce hand-built relationship synchronization logic in six maps:

SAP -> generic

generic -> SAP

PeopleSoft-> generic

generic-> PeopleSoft

Siebel-> generic

generic-> Siebel

However, if you use relationships in your solution, the relationship service provides prebuilt pattern implementations that maintains all these relationship instances for you.

Tools for working with relationships

The *relationship editor* in Integration Designer is the tool you use to model and design business integration relationships and roles. For detailed background and task information about creating relationships and using the relationship editor, see Creating relationships.

The *relationship service* is an infrastructure service in WebSphere ESB that maintains relationships and roles in the system and provides operations for relationship and role management.

The *relationship manager* is the administrative interface for managing relationships. It is accessed through the Relationship Manager pages of the administrative console.

Relationships can be invoked programmatically through the relationship service APIs.

Relationship service

The relationship service stores relationship data in relationship tables, where it keeps track of application-specific values across applications and across solutions. The relationship service provides operations for relationship and role management.

How relationships work

Relationships and roles are defined using the graphical interface of the relationship editor tool in Integration Designer. The relationship service stores the correlation data in tables in the relationship database in the default data source that you specify when you configure the relationship service. A separate table (sometimes called a participant table) stores information for each participant in the

relationship. The relationship service uses these relationship tables to keep track of the related application-specific values and propagate updated information across all the solutions.

Relationships, which are business artifacts, are deployed within a project or in a shared library. At the first deployment, the relationship service populates the data.

At run time, when maps or other WebSphere ESB components need a relationship instance, the instances of the relationship are either updated or retrieved, depending on the scenario.

Relationship and role instance data can be manipulated through three means:

- WebSphere ESB component Java snippet invocations of the relationship service APIs
- Relationship transformations in the WebSphere ESB business object mapping service
- The relationship manager tool

For detailed background and task information on creating relationships, identifying relationship types, and using the relationship editor, see Creating relationships topic.

Relationship manager

The relationship manager is the administrative interface for managing relationships. It is accessed through the Relationship Manager pages of the administrative console.

The relationship manager provides a graphical user interface for creating and manipulating relationship and role data at run time. You can manage relationship entities at all levels: relationship instance, role instance, and attribute data and property data levels. With the relationship manager, you can:

- View a list of the relationships in the system and detailed information for individual relationships
- Manage relationship instances:
  - Query relationship data to view subsets of instance data
  - Query relationship data to view subsets of instance data using database views
  - View a list of relationship instances that match a relationship query and detailed information about an instance
  - Edit the property values for a relationship instance
  - Create and delete relationship instances
- Manage roles and role instances:
  - View details about a role or a role instance
  - Edit role instance properties
  - Create and delete role instances for a relationship
  - Roll back relationship instance data to a point in time when you know the data is reliable
- Import data from an existing static relationship into your system, or export data from an existing static relationship to an RI or CSV file
- Remove relationship schema and data from the repository when the application that uses it is uninstalled

Relationships in Network Deployment environments

Relationships can be used in Network Deployment (ND) environments without any extra configuration.

In Network Deployment (ND) environments, relationships are installed in an application cluster. Relationships are then visible within the cluster, and all servers in the cluster have access to the instance data stored in the relationship database. The ability to run the relationship service in an ND environment makes it scalable and highly available.

The relationship manager allows relationships to be managed across different clusters through a centralized administrative interface. You connect the relationship manager to a server in a cluster by selecting its relationship MBean.

Relationship service APIs

Relationships can be invoked programmatically through the relationship service APIs, within or outside of business object maps.

Three API types are available:

- Relationship instance manipulation APIs (including create, update, delete instance data directly)
- Relationship pattern support APIs (including correlate(), correlateforeignKeyLookup)
- Relationship lookup patterns (lookup APIs)

Deployment architecture

The WebSphere ESB deployment architecture consists of software processes called servers, topological units referenced as nodes and cells, and the configuration repository used for storing configuration information.

Cells

In WebSphere ESB, *cells* are logical groupings of one or more nodes in a distributed network.

A cell is a configuration concept, a way for administrators to logically associate nodes with one another. Administrators define the nodes that make up a cell, according to the specific criteria that make sense in their organizational environments.

Administrative configuration data is stored in XML files. A cell retains master configuration files for each server in every node in the cell. Each node and server also have their own local configuration files. Changes to a local node or to a server configuration file are temporary, if the server belongs to the cell. While in effect, local changes override cell configurations. Changes to the master server and master node configuration files made at the cell level replace any temporary changes made at the node when the cell configuration documents are synchronized to the nodes. Synchronization occurs at designated events, such as when a server starts.

Servers

Servers provide the core functionality of WebSphere ESB. ESB servers extend, or augment, the ability of an application server to handle mediation modules. Other servers (deployment managers and node agents) are used for managing ESB servers.

An ESB server can be either a *stand-alone server* or a *managed server*. A managed server can optionally be a member of a *cluster*. A collection of managed servers, clusters of servers, and other middleware is called a *deployment environment*. In a deployment environment, each of the managed servers or clusters is configured for a specific function within the deployment environment (for example, destination host, application module host, or Common Event Infrastructure server). A stand-alone server is configured to provide all of the required functions.

Servers provide the runtime environment for mediation modules, for the resources that are used by those modules (data sources, activation specifications, and JMS destinations), and for IBM-supplied resources (message destinations and Common Event Infrastructure servers).

A *node agent* is an administrative agent that represents a node to your system and manages the servers on that node. Node agents monitor servers on a host system and route administrative requests to servers. The node agent is created when a node is federated to a deployment manager.

A *deployment manager* is an administrative agent that provides a centralized management view for multiple servers and clusters.

A stand-alone server is defined by a stand-alone profile; a deployment manager is defined by a deployment manager profile; managed servers are created within a *managed node*, which is defined by a custom profile.

Stand-alone servers

A stand-alone server provides an environment for deploying mediation modules in one server process. This server process includes, but is not limited to, an administrative console, a deployment target, the messaging support, and a Common Event Infrastructure server.

A stand-alone server is simple to set up, and has a First steps console from which you can start and stop the server and open the samples gallery and the administrative console. If you install the WebSphere ESB samples, and then open the samples gallery, a sample solution is deployed to the stand-alone server. You can explore the resources used for this sample in the administrative console.

You can deploy your own solutions to a stand-alone server, but a stand-alone server cannot provide the capacity, scalability, or robustness that is required of a production environment. For your production environment, it is better to use a network deployment environment.

It is possible to start off with a stand-alone server and later include it in a network deployment environment, by federating it to a deployment manager cell, *provided that no other nodes have been federated to that cell*. It is not possible to federate multiple stand-alone servers into one cell. To federate the stand-alone server, use the administrative console of the deployment manager or the **addNode** command. The stand-alone server must not be running when you federate it using the **addNode** command.

A stand-alone server is defined by a stand-alone server profile.

Clusters

Clusters are groups of servers that are managed together and participate in workload management.

A cluster can contain nodes or individual application servers. A node is usually a physical computer system with a distinct host IP address that is running one or more application servers. Clusters can be grouped under the configuration of a cell, which logically associates many servers and clusters with different configurations and applications with one another depending on the discretion of the administrator and what makes sense in their organizational environments.

Clusters are responsible for balancing workload among servers. Servers that are a part of a cluster are called cluster members. When you install an application on a cluster, the application is automatically installed on each cluster member.

Because each cluster member contains the same applications, you can distribute client tasks according to the capacities of the different machines by assigning weights to each server.

Assigning weights to the servers in a cluster improves performance and failover. Tasks are assigned to servers that have the capacity to perform the task operations. If one server is unavailable to perform the task, it is assigned to another cluster member. This reassignment capability has obvious advantages over running a single application server that can become overloaded if too many requests are made.

Profiles

A profile defines a unique runtime environment, with separate command files, configuration files, and log files. Profiles define three different types of environments on WebSphere ESB systems: stand-alone server, deployment manager, and managed node.

Using profiles, you can have more than one runtime environment on a system, without having to install multiple copies of the WebSphere ESB binary files.

Use the Profile Management Tool or the **manageprofiles** command-line utility to create profiles.

Note: On distributed platforms, each profile has a unique name. On the z/OS® platform, all profiles are named “default”.

The profile directory

Every profile in the system has its own directory containing all of its files. You specify the location of the profile directory when you create the profile. By default, it is in the `profiles` directory in the directory where WebSphere ESB is installed. For example, the Dmgr01 profile is in `C:\Program Files\IBM\WebSphere\ESB\profiles\Dmgr01`.

The First steps console

Every profile in the system has a First steps console. You can use this interface to familiarize yourself with the stand-alone server, deployment manager, or managed node.

The default profile

The first profile that you create within one installation of WebSphere ESB is the *default profile*. The default profile is the default target for commands issued from the `bin` directory in the directory where WebSphere ESB was installed. If only one profile exists on a system, every command operates on that profile. If you create another profile, you can make it the default.

Note: The default profile is not necessarily a profile whose name is “default”.

Augmenting profiles

If you already have a deployment manager profile, a custom profile, or a stand-alone server profile created for WebSphere Application Server Network Deployment, you can *augment* it to support WebSphere ESB in addition to existing function. To augment a profile, first install WebSphere ESB. Then use the Profile Management Tool or the **`manageprofiles`** command-line utility.

Restriction: You cannot augment a profile if it defines a managed node that is already federated to a deployment manager.

Deployment managers

A deployment manager is a server that manages operations for a logical group, or cell, of other servers. The deployment manager is the central location for administering the servers and clusters.

When creating a deployment environment, the deployment manager profile is the first profile that you create. The deployment manager has a First steps console, from which you can start and stop the deployment manager and start its administrative console. You use the administrative console of the deployment manager to manage the servers and clusters in the cell. This includes configuring servers and clusters, adding servers to clusters, starting and stopping servers and clusters, and deploying mediation modules.

Although the deployment manager is a type of server, you cannot deploy modules to the deployment manager itself.

Nodes

A *node* is a logical grouping of managed servers.

A node usually corresponds to a logical or physical computer system with a distinct IP host address. Nodes cannot span multiple computers. Node names usually are identical to the host name for the computer.

Nodes in the network deployment topology can be managed or unmanaged. A managed node has a node agent process that manages its configuration and servers. Unmanaged nodes do not have a node agent.

Managed nodes

A *managed node* is a node that is federated to a deployment manager and contains a node agent and can contain managed servers. In a managed node, you can configure and run managed servers.

The servers that are configured on a managed node make up the resources of your deployment environment. These servers are created, configured, started, stopped, managed and deleted using the administrative console of the deployment manager.

A managed node has a node agent that manages all servers on a node.

When a node is federated, a node agent process is created automatically. This node agent must be running to be able to manage the configuration of the profile. For example, when you do the following tasks:

- Start and stop server processes.
- Synchronize configuration data on the deployment manager with the copy on the node.

However, the node agent does not need to be running in order for the applications to run or to configure resources in the node.

A managed node can contain one or more servers, which are managed by a deployment manager. You can deploy solutions to the servers in a managed node, but the managed node does not contain a sample applications gallery. The managed node is defined by a custom profile and has a First steps console.

Unmanaged nodes

An unmanaged node does not have a node agent to manage its servers.

Unmanaged nodes in the Network Deployment topology can have server definitions such as Web servers , but not Application Server definitions. Unmanaged nodes can never be federated. That is, a node agent can never be added to an unmanaged node. Another type of unmanaged node is a stand-alone server. The deployment manager cannot manage this stand-alone server because it is not known to the cell. A stand-alone server can be federated. When it is federated, a node agent is automatically created. The node becomes a managed node in the cell.

Node agents

Node agents are administrative agents that route administrative requests to servers.

A node agent is a server that runs on every host computer system that participates in the Network Deployment configuration. It is purely an administrative agent and is not involved in application serving functions. A node agent also hosts other important administrative functions such as file transfer services, configuration synchronization, and performance monitoring.

Enterprise service bus messaging infrastructure

WebSphere ESB supports the integration of service-oriented, message-oriented, and event-driven technologies to provide a standards-based, messaging infrastructure in an integrated enterprise service bus.

The enterprise service capabilities that you can use for your enterprise applications provide not only a transport layer but mediation support to facilitate service interactions. The enterprise service bus is built around open standards and service-oriented architecture (SOA). It is based on the robust Java EE infrastructure and associated platform services provided by IBM WebSphere Application Server Network Deployment.

You can deploy WebSphere Enterprise Service Bus around your enterprise to extend the connectivity reach of the process integration solutions powered by separate installations of Process Server or other integration solutions as part of a federated ESB. For example, WebSphere Enterprise Service Bus can be installed closer to an SAP application to host an IBM WebSphere Adapter for SAP and to transform SAP messages before sending that information across the network to a BPEL process choreographed by Process Server.

Related concepts:

"About recovering the messaging subsystem" on page 1118
If the messaging system experiences problems you may need to recover the underlying messaging subsystem.

Messaging or queue destination hosts

A messaging or queue destination host provides the messaging function within a server. A server becomes the messaging destination host when you configure it as the messaging target.

A messaging engine runs within a server. The messaging engine provides messaging functions and a connection point for applications to connect to the bus. Service Component Architecture (SCA) asynchronous communication, JMS imports and exports, asynchronous internal processing use message queues on the messaging engine.

The deployment environment connects the message source to the message target through the bus when the application modules are deployed. Knowing the message source and message target helps you determine what type of deployment environment you need.

Applications can store persistent data in a data store, which is a set of tables in a database or schema, or in a file store. The messaging engine uses an instance of a JDBC data source to interact with that database.

Configure the messaging destination host when you define your deployment environment by using **Server** from the administrative console or designate the server as the destination host during software installation.

Data stores

Every messaging engine can use a data store, which is a set of tables in a database or schema that store persistent data.

All of the tables in the data store are held in the same database schema. You can create each data store in a separate database. Alternatively, you can create multiple data stores in the same database, with each data store using a different schema.

A messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine.

Data sources for WebSphere ESB

Data sources provide a link between applications and relational databases. The data sources that you use are affected by whether you set up a stand-alone environment or a network deployment environment.

Applications use a data source to obtain connections to a relational database. A data source is analogous to the Java EE Connector Architecture (JCA) connection factory, which provides connectivity to other types of enterprise information systems (EIS).

A data source is associated with a Java Database Connectivity (JDBC) provider, which supplies the driver implementation classes that connect with a specific type of database. Application components interact directly with the data source to obtain connection instances to your database. The connection pool that corresponds to each data source provides connection management.

You can create multiple data sources with different settings, and associate them with the same JDBC provider. For example, you might use multiple data sources to access different databases within the same database application. In WebSphere ESB, JDBC providers must implement one or both of the following data source interfaces. Use these interfaces to run the application in a single-phase or two-phase transaction protocol.

ConnectionPoolDataSource
: A data source that supports application participation in local and global transactions, except two-phase commit transactions. When a connection pool data source is involved in a global transaction, the transaction manager does not provide transaction recovery. The application is responsible for providing the backup recovery process if multiple resource managers are involved.

XADataSource
: A data source that supports application participation in any single-phase or two-phase transaction environment. When this data source is involved in a global transaction, the WebSphere Application Server transaction manager provides transaction recovery.

The following tables provide examples of typical stand-alone environment setups and typical deployment environment setups:

Table 16. Typical stand-alone environment setup

Datasource	Component	Scope	JNDI Name
WBI DataSource	CommonDB	Node	jdbc/WPSDB
SCA Application Bus ME data source	SCA ME	Server	jdbc/com.ibm.ws.sib/nlNode01.server1-SCA.APPLICATION.localhostNode01Cell.Bus
event	CEI	Server	jdbc/cei
CEI ME data source	CEI ME	Server	jdbc/com.ibm.ws.sib/nlNode01.server1-CEI.*cellName*.BUS

Table 17. Typical deployment environment setup

Datasource	Component	Scope	JNDI Name
WBI DataSource	CommonDB	Cell	jdbc/WPSDB
SCA Application Bus ME data source	SCA ME	Cluster	jdbc/com.ibm.ws.sib/clusterone-SCA.APPLICATION.enduranceTestCell01.Bus
event	CEI	Cluster	jdbc/cei
CEI ME data source	CEI ME	Cluster	jdbc/com.ibm.ws.sib/clusterone-CEI.*cellName*.BUS

JDBC providers

You can use JDBC providers to interact applications with relational databases.

Applications use JDBC providers to interact with relational databases. The JDBC provider supplies the specific JDBC driver implementation class for access to a specific type of database. To create a pool of connections to that database, you associate a data source with the JDBC provider. Together, the JDBC provider and the data source objects are functionally equivalent to the Java EE Connector Architecture (JCA) connection factory, which provides connectivity with a non-relational database.

Refer to the examples of both Typical stand-alone environment setup and Typical deployment environment setup in the previous topic.

For more information on JDBC providers, see "JDBC providers" in the WebSphere Application Server information center.

Service integration buses for WebSphere ESB

A service integration bus is a managed communication mechanism that supports service integration through synchronous and asynchronous messaging. A bus consists of interconnecting messaging engines that manage bus resources. It is one of the WebSphere Application Server technologies on which WebSphere ESB is based.

Some buses are automatically created for use by the system, the Service Component Architecture (SCA) applications that you deploy, and by other components. You can also create buses to support service integration logic or other applications, for example, to support applications that act as service requesters and providers within WebSphere ESB, or to link to WebSphere MQ.

A bus destination is a logical address to which applications can attach as a producer, consumer, or both. A queue destination is a bus destination that is used for point-to-point messaging.

Each bus can have one or more bus members, each of which is either a server or a cluster.

The *bus topology* is the physical arrangement of application servers, messaging engines, and WebSphere MQ queue managers, and the pattern of bus connections and links between them, that makes up your enterprise service bus.

Some service integration buses are created automatically to support WebSphere ESB. Up to three buses are created when you create your deployment environment or configure a server or cluster to support SCA applications. These buses each have five authentication aliases that you must configure.

SCA system bus

The *SCA system bus* is a service integration bus that is used to host queue destinations for Service Component Architecture (SCA) modules, such as mediation modules that provide service integration logic in the enterprise service bus (ESB). The SCA run time, which supports mediation modules, uses queue destinations on the system bus as an infrastructure to support asynchronous interactions between components and modules.

The system bus is automatically created when you create a deployment environment or when you configure a server or cluster to support SCA applications. The system bus provides a scope within which resources, such as queue destinations, are configured for mediation modules and interaction endpoints. The bus enables message routing between endpoints. You can specify the quality of service for the bus, including priority and reliability.

The bus name is SCA.SYSTEM.busID.Bus. The authentication alias used for securing this bus is SCA_Auth_Alias.

SCA application bus

The application bus destinations support the asynchronous communication of WebSphere Business Integration Adapters and other System Component Architecture components.

The application bus is automatically created when you create a deployment environment or when you configure a server or cluster to support SCA applications. The application bus is similar to service integration buses you might create to support service integration logic or other applications.

The bus name is SCA.APPLICATION.busID.Bus. The authentication alias used for securing this bus is SCA_Auth_Alias.

The Common Event Infrastructure bus

The Common Event Infrastructure bus is used for transmitting common base events, asynchronously, to the configured Common Event Infrastructure server.

The bus name is CommonEventInfrastructure_Bus. The authentication alias used for securing this bus is CommonEventInfrastructureJMSAuthAlias

Message Service clients

Message Service clients is available for C/C++ and .NET to enable non-Java applications to connect to the enterprise service bus.

Message Service Clients for C/C++ and .NET provides an API called XMS that has the same set of interfaces as the Java Message Service (JMS) API. Message Service Client for C/C++ contains two implementations of XMS, one for use by C applications and another for use by C++ applications. Message Service Client for .NET contains a fully-managed implementation of XMS, which can be used by any .NET compliant language.

You can obtain Message Service Clients for .NET from http://www-01.ibm.com/support/docview.wss?rs=0&q1=IA9H&uid=swg24011756&loc=en_US&cs=utf-8&cc=us&lang=en

You can obtain Message Service Clients for C/C++ from http://www-01.ibm.com/support/docview.wss?rs=0&q1=ia94&uid=swg24007092&loc=en_US&cs=utf-8&cc=us&lang=en.

You can also install and use the Java EE client support from WebSphere Application Server Network Deployment, including Web services Client, EJB Client, and JMS Client.

Chapter 3. Scenarios for Process Server and WebSphere Enterprise Service Bus

Scenarios for Process Server and WebSphere Enterprise Service Bus are intended to help you understand how to use the products to meet your business needs. Scenarios, and use cases, show how you can use the products together to solve a particular business need.

A scenario is a concise description of how you might use a product or group of products to achieve a goal. It can provide cross-component and multiproduct solutions, linking to tutorials, tasks and concepts in other sections.

Run time (or "Runtime product")
Run time or *runtime product* refers to WebSphere ESB or Process Server as the environment in which applications and services are run rather than developed.

Scenario
A *scenario* is a concise description of how a particular user might use a product, or group of products, to achieve a goal. It can provide cross-component and multiproduct solutions, linking to tasks and concepts in other sections.

Use case
A *use case* refers to a common user requirement or goal. For example, the requirement to connect service endpoints that use different protocols.

Related information:

PDF documentation
WebSphere Enterprise Service Bus documentation (in PDF format)

Information roadmaps
Business Process Management information roadmaps on IBM developerWorks organize information about WebSphere Process Server, WebSphere ESB, and the other products in the portfolio.

IBM Education Assistant
Multimedia educational modules about WebSphere ESB, provided by IBM Education Assistant.

Overview
Overview tab, on product library Web page. Use this page to access announcements, data sheets, and other general library documents related to WebSphere ESB.

Introduction: ESB scenarios and use cases

An enterprise service bus (ESB) is a key enabler for service-oriented architecture (SOA). This section shows some of the ways you can reuse your applications and services in a flexible and scalable way, using an ESB.

SOA is an architecture for building business applications from loosely coupled components, and for delivering a well-defined level of service.

Both WebSphere Enterprise Service Bus (WebSphere ESB) and WebSphere Process Server provide ESB functionality that enables you to build SOA solutions.

© Copyright IBM Corp. 2007, 2011

The scenarios and use cases follow the IBM service connectivity scenario (which is a scenario of use rather than a business scenario). The service connectivity scenario describes the way a middle tier (the ESB) provides routing and mediation, of messages passed between service requesters (consumers) and service providers. An ESB decouples the way a service requester views a service, from the implementation of a service. An ESB also decouples the business view of a solution from the technical aspects of service interactions; and allows integration and management of services. For more information on the IBM service connectivity scenario, see the: Service connectivity scenario.

Note: When the scenarios and use cases refer to mediation modules, they mean an SCA module that can change the format, content, or target of service requests. Generally, a mediation module contains one or more mediation flow components. However, from Version 6.2, WebSphere Business Process Manager business modules can also contain mediation flow components. Therefore, when the documentation refers to mediation modules, this is for simplicity: the mediation functionality could also be found in business modules.

Patterns

You can create an ESB with the help of patterns. There are many types and levels of patterns; the following are particularly relevant to an ESB and to SOA.

- The ESB pattern. An ESB is a high-level, architectural, pattern in its own right. The ESB pattern can involve several different roles. For example, adapter, hub, gateway and registry. Some products can perform more than one of those roles. WebSphere ESB and Process Server can act as a gateway and a hub, and as an adapter. They use WebSphere Service Registry and Repository (WSRR) to provide the registry role.
- Standard enterprise integration patterns. Individual interactions exposed by an ESB can use low-level design patterns such as mediation. Mediations are created in mediation modules, using IBM Integration Designer. IBM Integration Designer uses patterns, embodied in building blocks called mediation primitives. Mediation modules are deployed to the runtime product.
- Patterns for e-business. An ESB can be part of a wider solution pattern. IBM provides e-business patterns that include ESB functionality.

SOA considerations

There are certain key considerations when building a service-oriented architecture (SOA) solution. This section explains some of the key goals of an SOA solution, and how they apply to an enterprise service bus (ESB).

Introduction

SOA is an architecture for building business applications from loosely coupled components, and for delivering a well-defined level of service. Decoupling service requesters (consumers) from service providers greatly increases flexibility. It allows the substitution of one service provider for another, without the requester being aware of the change and without the need to alter the architecture.

You can use an SOA to connect across different systems without undue complexity, generally using an ESB. You can use an ESB to reuse assets in a flexible way, by passing service interactions through logic called mediations. Mediations operate on messages in transit between requesters and providers; they allow facilities such as message filtering, protocol conversion, database logging, and dynamic routing.

WebSphere details

All SOA solutions have a similar life cycle, which for WebSphere ESB and WebSphere Business Process Manager can be summarized as follows:

1. Model.
   - Capture your business design. This includes relating your business processes to the services they will need.
2. Assemble.
   - Integrate your existing applications, databases, and services. Use WebSphere Integration Developer to create a mediation module that integrates your existing assets, and defines qualities of service.
3. Deploy.
   - You might need to create the runtime environment before deploying your mediation module to the run time. Creating the runtime environment includes resolving resource dependencies and integrity constraints.
4. Manage.
   - Use the runtime administrative console to carry out routine maintenance such as administering and securing applications, resources, and users. This phase also includes monitoring the availability of applications and systems, and monitoring the performance of service requests and responses. The run time provides facilities for monitoring and tuning, but you might also use monitoring products such as IBM Tivoli Composite Application Management for SOA.

Flexibility and reuse

With a service-oriented architecture (SOA) you can reuse existing resources, and integrate with external systems. This section covers the concepts of flexibility and reuse in an SOA, and how an enterprise service bus (ESB) enables these concepts.

Introduction

An SOA enables services to be consumed more effectively, so you can respond faster to business needs. By working with common interfaces, you can more rapidly integrate external systems into an existing environment.

Planning

When you are planning how to reuse services in a flexible way, consider the following factors:

- The addresses of service endpoints can be stored in a number of different ways. You can configure endpoints when you assemble your module or, for more flexibility, you can store endpoint information in a registry or a database.
- More than one service might be able to fulfil a service request. If you use a registry, you can get a list of services that meet certain criteria.
- The criteria for routing messages can be simple, or complex. Messages can be routed based upon one factor, such as a customer identifier or a customer type, or upon multiple factors.
- Different services often need different message formats. The order of fields might need to be different, or new fields might need to be added. With an ESB you can alter the format of both the message body and the message header.

Mediations

An ESB enables reuse by mediating between service requesters (consumers) and service providers. Mediation covers the following functions:

- Content-based routing
- Transformation
- Validation
- Enrichment
- Service invocation

WebSphere ESB and WebSphere Process Server

A mediation flow encapsulates logic that is executed when a message is received by WebSphere ESB or WebSphere Business Process Manager. The integration developer specifies the interfaces of the service requester and service provider, so the mediation flow knows what kinds of messages it will process.

If you are using WebSphere ESB, you can implement a mediation flow in a mediation module. If you are using WebSphere Business Process Manager, you can implement a mediation flow in a mediation module or a module. Both a mediation module and a module are types of Service Component Architecture (SCA) module, and both can be referred to as service modules. The mediation flow is created using WebSphere Integration Developer, and the resulting service module is deployed to the run time in an enterprise archive (EAR) file.

A service module can have exports that define interactions with service requesters, and imports that define interactions with service providers. Each import and each export has a binding that defines the actual connection mechanism. For example, SOAP/HTTP. When you use dynamic routing you might not need an import, except to provide a default endpoint.

You can put the business objects and interfaces that are used by imports and exports into libraries, so that they can be shared and reused. Any service module that wants to use a library is defined to have a dependency on that library.

Mediation primitives

A service module can contain one or more mediation flow components. Mediation flow components host mediation flows, which are assembled from mediation primitives. Mediation primitives are the building blocks of mediation flows and each mediation primitive lets you do different things with a message.

Loosely bound modules: databases and registries

If you store service addresses in a database or registry you can change which services get invoked by updating the contents of your database or registry; or by pointing your mediation flow at a different registry instance or database table.

Generally, if you want to change the database your mediation flow uses, the integration developer must update the mediation primitive properties and create a new EAR file, which you deploy to the runtime. Alternatively, the integration developer can promote certain mediation primitive properties (make the properties visible at run time) and the runtime administrator can then update the properties without needing to redeploy the EAR file. For example, the Database Lookup

mediation primitive allows you to promote the **Table name** property. The **Table name** property contains the name of the database table, including the schema name. For example, `myschema.mytable`.

Storing service addresses in a registry can make a solution even more flexible and reusable, because a mediation flow can extract multiple service addresses for one type of business service. Alternatively, a registry could return one service address based upon more detailed criteria. For example, the registry might return services in a certain geographical or linguistic area.

Patterns

The ESB architectural pattern has a number of common instantiations that can be represented as design patterns. Some design patterns can help increase the flexibility of your systems, and the reuse of your service.

This topic describes two of the common patterns used for routing.

Pattern one: using one service result to invoke another service

Invoke a service in the middle of a mediation flow, and use the result to make a routing decision.

You can use the Service Invoke mediation primitive to call an external service in the middle of a mediation flow. You can determine the characteristics of the call to the service by customizing the properties of the mediation primitive. (If you need additional mediation primitive flexibility, you can use the Custom Mediation primitive, rather than the Service Invoke mediation primitive.)

You can use the result of invoking the service to do further processing, and make a routing decision. You could also use the result of invoking the service to augment the original request message.

Generally, you need to save the current message before making a service call that is in the middle of a mediation flow. For example, you could use the XSLT mediation primitive to save the current message in the transient context. You could then make the service call, and afterward use another XSLT mediation primitive to make use of the service response and the original message, which is saved in the transient context.

Pattern two: using registries and databases to store service endpoints

Invoke a service by querying a registry or database for the service endpoint.

You can use the following mediation primitives to directly call a registry or a database:

- The Endpoint Lookup mediation primitive. You can determine the characteristics of the call to the registry by customizing the properties of the mediation primitive.
- The Database Lookup mediation primitive. You can determine the characteristics of the call to the database by customizing the properties of the mediation primitive.

If you need additional mediation primitive flexibility, you can use the Custom Mediation primitive.

You can select a service endpoint from an external source, based on a condition or property in your flow. At development time:

- You need to know the port type, or interface, of the service you are invoking.
- You do not need to know the address of the service you are invoking (you get the address at run time from a registry or a database).

Choosing which service endpoint to use at run time, rather than integration time, enables services to be more loosely coupled, and businesses to react faster to change. New services can be added without touching the modules that have been deployed to the run time.

You can use the IBM WebSphere Service Registry and Repository (WSRR) to store service endpoint information, and use the Endpoint Lookup mediation primitive to query the registry for a service endpoint, or a set of service endpoints. If you retrieve a set of endpoints, you can use another mediation primitive to select the one you want to use. Alternatively, you can exploit additional endpoints as alternative targets, to be used if the invocation of the first service fails for some reason.

Figure 53. Dynamic routing using the Endpoint Lookup mediation primitive

Alternatively, you can use a database to store service endpoint information, and retrieve the service endpoint information using the Database Lookup mediation primitive.

To enable dynamic routing, with or without a registry, you must ensure that the following conditions are satisfied at integration time:

- The mediation flow stores the dynamic endpoint address at the following location in the message:`/headers/SMOHeader/Target/address`.

  The Endpoint Lookup mediation primitive can automatically copy an endpoint to the correct part of the message (SMO). However, you can use other mediation primitives to copy the dynamic endpoint to the correct location.
- The **`Use dynamic endpoint`** property is set. The **`Use dynamic endpoint`** property is set, by default.

Connectivity

A service-oriented architecture (SOA) must enable integration of both new and existing services, and this integration requires connectivity to a range of different systems. This section covers the concept of connectivity in an SOA, and how an enterprise service bus (ESB) enables this concept.

Introduction

An ESB links together many types of data, applications, protocols and platforms, allowing you to adapt quickly to changes in your business environment and information technology (IT) systems. Having the ability to connect to many types of system promotes reuse: your data and business logic become reusable, and applications are easier to service-enable.

How to enable connectivity for WebSphere ESB and IBM Business Process Manager

To enable ESB connectivity you need to perform the following tasks:

1. Identify, or create, your services.
2. Create a module that makes use of your services. Use IBM Integration Designer to create a mediation module.
3. Define resources for the deployment environment. You can use the runtime administrative console (or scripts) to define, or configure, many of your resources.
4. Deploy your mediation module to the deployment environment. You can export your mediation module from IBM Integration Designer as an EAR file, and install the EAR file on the runtime system.

How to enable connectivity at integration development

To enable ESB connectivity, use IBM Integration Designer to specify:

- The services you want to interact with (exports and imports).
- The format of the data to be exchanged (interfaces).
- The protocol of the service you want to interact with (bindings).

You must provide interfaces to the services you want to use. Generally, service interfaces for the ESB are defined using WSDL portTypes. Having a formal (WSDL) description enables message exchanges to be viewed as service calls. The service requester (consumer) sends a service request to the ESB, and the service provider receives a service request from the ESB. Both the export and the import need an interface definition that describes the structure of the data that is exchanged.

You can use IBM Integration Designer to create interfaces, and generate WSDL. For example, to service-enable an application you might use the business object editor to describe the content of the message you want to send, and then use the business object when you create an interface. You could then specify the interface on the export and import, and select the precise binding. The binding specifies both the protocol and the physical format used to send and receive service messages.

Protocol transformation

After a binding has been configured, protocol transformation is handled by the run time.

If two services have the same interface (as defined by their WSDL) and no mediations are required, you can use IBM Integration Designer to wire exports directly to imports. If two services have different interfaces, or you need to mediate the message, a mediation flow component is needed to do the transformations.

How to enable connectivity at run time

There are aspects of connectivity that you have to define outside of the integration environment. At run time, your mediation modules might need access to databases, registries, messaging systems, and enterprise information systems (EIS). You can use the runtime administrative console, or scripts, to define or configure the following resources to the run time.

- Messaging systems.
- EIS systems. IBM adapters allow you to connect to EIS systems.
- Registries. To define registries to the run time use the administrative console. Click **Service integration** > **WSRR definitions** > **New**. For more information, see: Creating a new WSRR definition
- Databases. You can connect to databases in various ways. For example, you can connect to a database:
  - As the service endpoint. IBM adapters allow you to connect to EIS systems, including databases.
  - From inside a mediation module. Some mediation primitives are designed to make use of databases. For example:
    - The Database Lookup mediation primitive can change messages using information from a user-supplied database.
    - The Message Logger mediation primitive uses a database for logging, and this mediation primitive can make use of the Common database that is created when you install the runtime product.

Types of service requesters and providers you can connect to

The following is a summary of the types of service requesters and service providers that you can connect to, through the ESB. Some systems naturally enable asynchronous communication, while others fit better with synchronous communications.

Asynchronous communications, using messaging queues, can have the following SOA advantages:

- A solution design based on asynchronous messaging can allow communicating partners to be decoupled, helping towards the flexibility goal of SOA.
- Asynchronous messaging and queueing allow you to set levels of reliability, which can help you move towards the robustness goal of SOA.

You might choose to use a Java Message Service (JMS) provider, or IBM WebSphere MQ, depending on what systems you currently have and what systems you might connect to in the future.

Web services

You can connect to Web services using a Web service binding.

IBM Integration Designer provides two kinds of bindings that are suitable for Web services: Web service bindings and HTTP bindings.

The Web service binding, which can be either SOAP/HTTP or SOAP/JMS, is simpler and follows established standards for remote execution. The binding is described in a WSDL file, so it works naturally with the WSDL interfaces that are used with most components and exports.

If you want to share Web service bindings between modules deployed in the same cluster, you need to share the port.

At run time, you can specify a new endpoint from the administrative console by modifying the binding. For more information see: Changing a Web service import binding

Messaging systems

You can connect to external messaging systems using a JMS binding. JMS can exploit various transport types, including TCP/IP and HTTP or HTTPS.

The following JMS bindings are supported:

- Generic JMS 1.1 compliant providers (third-party JMS providers), using generic JMS bindings. Generic JMS bindings support JMS providers that support the Application Server Facility of the JMS 1.1 specification, but do not support JCA 1.5. Generic JMS 1.1 compliant providers are provided by: Oracle AQ, TIBCO, SonicMQ, WebMethods, BEA WebLogic, and WebSphere MQ.
  - Due to its generic nature, there are no provider-specific connectivity options, and the generic JMS binding has limited capability to generate resources at deployment and installation.
  - You must set up connectivity to and from a third-party JMS provider in order to use the generic JMS binding. For more information, see: Setting up connectivity for the Generic JMS binding
- WebSphere Application Server default messaging provider, compliant with JMS JCA 1.5. If you want to use the WebSphere Application Server default messaging provider you specify JMS bindings.
  - When you generate JMS bindings using IBM Integration Designer, you have the option of creating the provider resources when the mediation module is deployed. Alternatively, you can specify the Java Naming and Directory Interface (JNDI) name of the runtime resources your JMS imports and exports must use. Runtime configuration of the JMS binding depends on which option you select at development time.
  - At run time, you can configure JMS import and export bindings to apply special features of the resource, using the administrative console. For more information, see: Viewing and updating JMS bindings
- WebSphere MQ JMS provider. If you want to connect to WebSphere MQ using its JMS provider you specify WebSphere MQ JMS bindings. The WebSphere MQ JMS binding is designed to interoperate with JMS applications deployed to WebSphere MQ, and to expose messages according to the JMS message model.

If you invoke a mediation module by sending a JMS message to an export with a JMS binding, you can use any part of the message inside the mediation flow component to make mediation decisions, such as routing or transformation.

WebSphere MQ

You can connect to native WebSphere MQ applications using WebSphere MQ bindings.

When you generate WebSphere MQ bindings using IBM Integration Designer, you have the option of creating the provider resources when the mediation module is deployed. Alternatively, you can specify the Java Naming and Directory Interface (JNDI) name of the runtime resources your WebSphere MQ imports and exports must use. Runtime configuration of the WebSphere MQ binding depends on which option you select at development time.

Note: the resources created do not include the queue or the queue manager, which the WebSphere MQ administrator must create if they do not already exist.

You can connect to a WebSphere MQ service provider using communications that are one-way or two-way (request-response). For two-way communications, A WebSphere MQ request-reponse application can use various techniques to correlate response messages with requests. For more information on WebSphere MQ bindings, see: WebSphere MQ bindings

Before you can run an application containing WebSphere MQ bindings, you must ensure that the MQ_INSTALL_ROOT environment variable is set to a supported version of WebSphere MQ.

At run time, you can use the administrative console to tune, or set, special features of the resources. For more information, see: Viewing and updating WebSphere MQ bindings

EIS systems

IBM adapters allow you to create integrated processes that include EIS systems.

Adapters service-enable external applications and data storage facilities by connecting them to the ESB, which powers your SOA. Adapters are sometimes referred to as resource adapters, and provide a standard interface to proprietary systems. Using standard interfaces avoids the maintenance issues associated with non-standard solutions.

There are two types of IBM adapters:

- WebSphere Adapters, also referred to as JCA adapters.
  - WebSphere Adapters are based on Java EE Connector architecture (JCA), and are the recommended adapters to use with your ESB.
  - If an application is developed with a WebSphere Adapter embedded, the adapter is deployed with the application.
- WebSphere Business Integration Adapters.
  - WebSphere Business Integration Adapters reside outside of the ESB. The run time communicates with this type of adapter through a Java Message Service (JMS) transport layer.

Both types of adapter can be split into two classes:

- Technology Adapters
- Application Adapters

Service Component Architecture (SCA) modules

When you are designing a system, or developing modules, you might want to link SCA modules together. You can do this if you are using the default (SCA) bindings. Mediation modules are types of SCA modules.

If you want to link two SCA modules, the first module has an import with an SCA binding, and the second module has an export with an SCA binding. At run time, you can change which SCA modules are called by changing the SCA import binding. For more information on changing SCA import bindings, see: Changing an SCA import binding

Governance

A service-oriented architecture (SOA) allows you to govern your processes and services in a secure and efficient way. This section covers the concept of governance in an SOA, and how the ESB enables this concept.

Introduction

In the business world any key asset needs to be carefully managed in order to get the maximum business benefit. In other words, key assets need governance.

The life cycle of a service in an SOA requires governance processes. Governance processes ensure that standards and policies are enforced, and that change occurs in a controlled fashion and with the appropriate authority.

Governance covers the following areas:

- Aligning business goals with information technology goals
- Managing resources
- Managing risk
- Managing performance
- Reducing costs

WebSphere Service Registry and Repository (WSRR)

The scope of governance in an SOA environment extends beyond any single product. However, WSRR is a powerful enabler for SOA governance, and provides a number of mechanisms that you can use to implement the governance processes that have been specified by your business design.

WSRR can be used with the ESB, and provides the following governance capabilities:

- Business governance. Business governance ensures that your business processes are legal, auditable, and correctly mapped to information technology services. Business governance can include the following features:
  - Policy enforcement and change auditing, using a simple policy life cycle, and status metadata, in the registry.
  - Security to control who can change policies and who can register service providers. WSRR gives you security-based and role-based access control, to metadata managed in the registry.
- Information technology governance. Information technology governance ensures that updated systems perform as intended, and that no other systems are affected. Information technology can include the following feature:
  - Life cycle management allows you to categorize services as being in different states. For example, model, assemble, deploy, and manage. Life cycle status is held in the registry. You can use the life cycle status to filter the visibility of services, according to usage and context. You can define a life cycle that includes the categories required by your organization.

Using WSRR with the ESB

In order for the run time to make use of an existing WSRR registry, you need to take the following steps:

1. Create and configure an SCA module that uses the registry.

   The integration developer creates a mediation flow that contains an Endpoint Lookup mediation primitive or a Policy Resolution mediation primitive, and exports the module in an enterprise archive (EAR) file.

   Note: The governance state is specified on the **`Classification`** property of the mediation primitives.
2. Add the appropriate service endpoints, or mediation policies, to the registry.

   The WSRR administrator loads documents (for example, WSDL or SCA module documents) for the services and policies you want to use.
3. Make the appropriate WSRR objects governed.

   After the WSRR administrator has made an object governed, the object can be moved from one life-cycle state to another. This means that you can store both production level and test level objects in one registry, or store production level versions in the same registry.
4. Optional: Configure the WSRR entities.

   The WSRR administrator might need to take other actions, depending on what the integration developer has created. For example, on the Endpoint Lookup primitive there might be user-defined properties; these properties let mediation flows retrieve service endpoints that meet specific conditions. In this case, the WSRR administrator would need to create custom properties with a key and a value.
5. Create and configure a registry definition in the runtime environment.

   The runtime administrator creates a WSRR definition so that the run time knows which instance of WSRR to access. If you are using a secured WSRR, the runtime administrator needs to specify an authentication alias.
6. Deploy the SCA module.

   The runtime administrator installs the EAR file containing the mediation flow.

Security

A service-oriented architecture (SOA) allows the loose coupling of services, and therefore requires careful attention to security. This section introduces the IBM SOA security reference model and how this relates to an enterprise service bus (ESB).

Introduction

Security is particularly important for implementations structured according to SOA principles, because of the loose coupling of services and applications. Operations can occur across trust boundaries.

When using an ESB it is important to protect business information and establish business trust relationships for identity and data. Because requests can come from external entities and cross security domains, the propagation of identity across domains is a key consideration.

An ESB gateway can manage the trust relationship between the service requester and service provider; it can also make use of numerous security services, including those responsible for:

- Message level security.
- Confidentiality and integrity.
- Identity and authentication.
- Authorization and privacy.
- Federation of identities between requester and provider.

This section introduces some of the concepts of the IBM SOA security reference model.

Security in the SOA life cycle

Every stage in the SOA life cycle requires planning for security, as shown in the following diagram.

Figure 54. The SOA life cycle with security

The ESB can make use of various security techniques.

IBM SOA security reference model

The IBM SOA security reference model provides an overview of the components of SOA security. There are three main components to the security reference model:

Figure 55. IBM SOA security reference model

- IT security services.
  - The building blocks to provide security functions as services. The use of common IT security services enables a consistent security implementation; and reduces the cost of developing and deploying services. IT security services can be used by different components in the SOA environment. For example, gateways, proxy servers, application servers, data servers, and operating systems.
- Security policy infrastructure.
  - How processes and services manage the behavior of the underlying infrastructure, in order to secure access to information, specify information availability and provide for audits.
- Business security services.
  - A collection of services and capabilities required to provide a verifiably-secure deployment environment for an SOA solution.

IT security services

The IT security services are made up of the following components:

Figure 56. IT security services

- Identity Services.
  - Identity foundation.

- There can be multiple user repositories. For example, there might be an enterprise user repository for identity information that is common to many applications, and individual user repositories for each of the applications. The identity of a customer can be spread across the repositories.

– Identity provisioning.
  - Enable a policy-based approach to managing identities across user repositories. You can use a central provisioning service to create, modify, and delete identity information across user repositories.

– Identity federation.
  - Translate secure identity information to a different format.
    - For example, a service request might contain a user name security token. An ESB mediation module could call a trust service that translates the security token to a format suitable for the service provider. The mediation module could also perform identity mapping, to translate identity and attribute information. If the incoming user name was `jsmith`, it might be translated to `jsmith1234`.

.

- Authentication Services.
  – Portal users can be requested to present authentication credentials as a prerequisite to service authorization. Authentication credentials include: a user name and password, hardware token, or biometrics. Authentication credentials are passed to the authentication service, which verifies the identity of the user.
- Authorization and Privacy Services.
  – Requests must be authorized before being granted access.
    - For example, a mediation module could call an authorization and privacy service to implement service level authorization (authorization that determines who can call a service). The advantage of using the ESB is that it can perform authorization for a service provider before making any request to the service provider. There might be finer grained authorization at the application level, and further authorization at the data level (often implemented by a database).
- Confidentiality and Integrity Services
  – Messages, data stores and systems need to be secured to prevent unauthorized access. User registries and service provider databases have to be protected, as well as the systems on which they reside.
    - The ESB provides support for the following data privacy and integrity solutions. For more information, see: Data integrity and privacy
      - Point to point protection. Normally, the whole data stream is protected at the protocol level, rather than the message level. Secure Sockets Layer (SSL) is the most common example of a point to point protection scheme.
      - Message level protection. WS-Security adds support for authentication, integrity, and privacy for a single message; and is suitable for securing messages in an asynchronous environment.
- Audit services.
  – You can create an audit event for any of your security events.
- Non-repudiation services.
  – Non-repudiation services provide evidence that a message flow has taken place. From the point of view of the service provider, the ESB receives the initial service request, and calls the service provider. The ESB can invoke the non-repudiation services, based on the incoming request, and the service

provider can access the non-repudiation services to record the message coming from the ESB. From the point of view of the service requester, the important evidence to record is that the ESB has responded to the request. Because access to the service provider is not direct, evidence for the non-repudiation record might be based on the secure channel between requester and the ESB. The service requester can create a non-repudiation record by using the audit service to record the signed data response, with a time stamp to indicate when the response was received.

Security policy infrastructure

The Security policy infrastructure is made up of the following components:

Figure 57. Security policy infrastructure

- Policy administration.
  - Policy administration deals with the life cycle of security policies. It can sub-divided into message protection policies and provider policies.
    - Message protection policies specify the interaction between service requesters and the ESB, and between service providers and the ESB. For example, if the ESB is expecting signed and encrypted messages from a service provider, the service provider must be aware of this so that outbound messages are correctly prepared.
    - Provider policies are policies that are internal to a particular service provider. For example, authorization policies are often used to describe the entities able to access a service.
- Policy distribution and transformation.
  - High-level, abstract policies need to be made available to components that require the information specified in those policies. For example, a centrally-defined security policy for accessing application functions could get transformed and distributed to include the following:
    - Web services security management policy distributed to a WebSphere DataPower® security appliance.
    - Permission set to RACF® on z/OS.
- Policy decision and enforcement.
  - Policy decision points can include:
    - Common audit services that evaluate their policy for the generation and distribution of audit data. For example, a central data warehouse might be used in the enterprise to store audit records collected across solution components.
  - Policy enforcement points are usually distributed across multiple components in the SOA environment. For example, an ESB can enforce policies relating to the protection of messages, based on the results of cryptographic operations for signature validation or decryption.
- Monitoring and reporting.

Robustness

There are many ways in which a service-oriented architecture (SOA) solution can be made more robust. This section covers some of the ways to increase robustness in an SOA.

Introduction

There are numerous ways in which an SOA solution can be made more robust. For example:

- Failover support. When a server fails, an application can continue to process work on other servers in the cluster.
- Persistent messages. Persistent messages are recovered if a failure occurs.
- Persistent events. You can use the common event infrastructure (CEI) to persistently store and retrieve events.
- Transactions. Transactions can be a key enabler because they ensure end-to-end data integrity across multiple resources.

Failover support

A clustered environment provides the following SOA-related benefits:

- Failover support. When a server fails, an application can continue to process work on other servers in the cluster. This allows recovery efforts to proceed without affecting the application users.
- Maintainability. You can stop a server for planned maintenance without stopping application processing.
- Flexibility. You can add or remove capacity as needed, by using the administrative console of the deployment manager.

Persistent messages

Messaging systems can be used to send persistent messages.

WebSphere MQ and other Java Message Service (JMS) providers support the following types of message:

- Persistent messages. Persistent messages are guaranteed to be successfully consumed only once. Messages are not lost.
- Non-persistent messages. Non-persistent messages are guaranteed to be delivered at most once. Messages can be lost

Persistent messages have a performance impact; therefore, use persistent messages for essential data only.

Persistent Events

CEI provides facilities for the runtime environment to persistently store and retrieve events.

You can use the Event Emitter mediation primitive to send out business events, during a mediation flow. The events are generated in the form of Common Base Events and are sent to a CEI server.

Transactions

Transactions ensure end-to-end data integrity across multiple resources. A robust SOA solution should consider data integrity at all levels.

Introduction

A transaction is a unit of activity. Within a transaction, multiple updates to resources can be made an indivisible unit of work: either all the updates are made permanent, or none of the updates are made permanent.

If a transaction involves multiple resource managers, for example multiple database managers, then an external transaction manager is needed to coordinate the individual resource managers. A transaction that spans multiple resource managers is called a global, or distributed, transaction. WebSphere Application Server is a transaction manager that can coordinate global transactions, and the runtime makes use of the WebSphere Application Server functionality.

Planning

When planning how to use transactions, consider the following:

- If you use Web Services Atomic Transaction (WS-AT) to propagate transactional context from the service requestor, then you might want to make the mediation flow part of the service requester's transaction.
- When developing your mediation flow, in IBM Integration Designer, the default is that the mediation is not part of a transaction. Therefore, the mediation does not propagate any transactional context provided by the service requester.
- Different import and export bindings have different transactional capabilities. A Web services binding can propagate a transaction from the service requester. However, a JMS binding cannot propagate a transaction from the service requester.
- WebSphere Business Integration Adapters are based on JMS. Therefore, as soon as a message is put on a message queue it is considered complete, from a transaction point of view. If a messaging system is defined not to time out, and the ESB messages are permanent, then the messaging system ensures that the messages are delivered.
- JCA adapters allow full transactional updates.

Creating a transaction for a Web services request

If the service requester and provider both have Web services bindings, and the service requester is using WS-AT, you can make your mediation flow part of the global transaction. Use IBM Integration Designer to define the following quality of service qualifiers:

- Set the implementation's *Transaction* qualifer to `Global` or `Any`.
- Set the mediation modules's interface *Join transaction* qualifer to `True`.
- Set the mediation modules's import interface *Join transaction* qualifer to `True`.

Figure 58. Transaction covering service requester, mediation flow and service provider

.

Controlling individual parts of a mediation flow

Some mediation primitives have a transaction property, so you can control whether a particular aspect of a flow takes part in a global transaction. For example, there might be a business requirement to log certain events, whether or not the service request completes successfully or not. In this case you might decide to keep logging, or the emission of events, outside of the main transaction.

Both the Message Logger mediation primitive and the Event Emitter mediation primitive have a *Transaction mode* property. If the integration developer `promotes` these properties, they are visible at runtime and can be updated by the runtime administrator without needing to redeploy the EAR file. By default, a Message Logger mediation primitive is part of any global transaction that you define and, by default, an Event Emitter mediation primitive uses the default transaction setting in the CEI emitter. The CEI Emitter is a library that provides support for applications that report events.

Management

A service-oriented architecture (SOA) helps you to manage services more easily. This section covers the management of Service Component Architecture (SCA) modules that invoke your services.

Introduction

With an enterprise service bus (ESB) you can manage your services more easily, so you can respond faster to changing business needs. You can manage the SCA modules that invoke your services, from the administrative console, from the command line, and from scripts.

With an ESB you can mediate between service requesters (consumers) and service providers. If you use WebSphere ESB, mediation flows occur in a type of SCA module that IBM Integration Designer calls a mediation module. If you use WebSphere Process Server, mediation flows can occur either in mediation modules

or in a type of SCA module that IBM Integration Designer simply calls a module. The administrative console refers to both mediation modules and modules as SCA modules.

This section deals primarily with the management of SCA modules that contain mediation flows. Mediation primitives are the building blocks of mediation flows and each mediation primitive lets you do different things with a message.

Note: It is often the case that WebSphere Process Server modules do not contain mediation flows.

Managing SCA modules

After you have deployed an SCA module to the run time, you might want to do a number of tasks to manage your service flows. The following information covers some of the tasks.

Enabling service connectivity flows

Each SCA module has an associated application. In order for service connectivity flows to be enabled, the application has to be started. By default, the application starts automatically when the server starts. However, you might want to enable the service connectivity immediately after you deploy your mediation module.

Use the administrative console to start the application associated your with your SCA module. For more information, see: Starting and stopping service modules

Changing SCA module properties without redeploying your module

IBM Integration Designer allows you to make some SCA module properties visible to the runtime administrator. SCA module properties that are visible to the administrator are called **promoted properties**.

IBM Integration Designer lets you give alias names to promoted properties; and it is the alias names that are displayed on the administrative console. For more information, see: Changing the value of mediation flow properties at runtime

You can use the administrative console to change the value of promoted properties without having to redeploy an SCA module, or restart the server or module. However, any changes that are made at runtime do not affect the integration development version of the module.

Mediation flows start to use the value of a changed property as soon as the configuration update is visible to their server. In a network deployment environment this occurs after a node's synchronization of configuration data with its Deployment Manager. For more information, see: Changing service module properties

Changing service endpoints

At run time, there are various ways in which you can change the service endpoint that an SCA module invokes.

For example, suppose you have a mediation module, Transform, that does message enhancement and transformation, and invokes another mediation module, Route,

to do message routing. If you want the Transform module to call a different routing module, you can do this in the runtime environment if the mediation modules connect to each other using the default (SCA) bindings.

Any changes that you make to an SCA module at run time do not affect the integration development version. Therefore, if you want to change an SCA module permanently you might typically change it in the integration environment, and redeploy the SCA module.

The following examples describe some of the ways you can change service endpoints at run time:

- If you have an SCA module that invokes another SCA module, using SCA bindings, you can use the administrative console to change which SCA module is invoked: **Applications** > **SCA Modules** > **Imports** > **Binding**. For more information, see: Changing an SCA import binding
- If you have an SCA module that directly invokes a Web service, you can use the administrative console to change the Web service that is invoked: **Applications** > **SCA Modules** > **Imports** > **Binding**. For more information, see: Changing a Web service import binding
- If you have an SCA module that gets its service endpoints from a database, you can determine which services get invoked by updating the database.
- If you have an SCA module that gets its service endpoints from a registry, you can determine which services get invoked by updating the registry or by updating the query that is used. You can only use a registry to provide certain types of service endpoints. For more information, see: Endpoint Lookup mediation primitive

Changing the registry used by an SCA module

At run time, you can change which WebSphere Service Registry and Repository (WSRR) instance is used by your mediation flow.

If the Endpoint Lookup mediation primitive (used in your SCA module) does not specify a registry, the mediation flow makes use of the default WSRR definition. If the integration developer has promoted the **`Registry Name`** property, you can check to see if the property has a value.

You can change the default WSRR definition: **Service integration** > **WSRR definitions**, and set which WSRR definition you want to be the system default. For more information, see: Setting a default WSRR definition

If the integration developer has promoted the **`Registry Name`** property of the Endpoint Lookup mediation primitive, you can set the **`Registry Name`** value from the administrative console. You could set a particular SCA module to use a particular registry instance.

Validating messages

When designing your service flows, it might be a requirement to ensure that the message type at run time matches the message type definition. You can validate messages against their schema by using mediation primitives that have a **`Validate input`** property, for example, the XSLT and Message Element Setter mediation primitives.

Because message validation can affect performance, the integration developer would normally disable the **Validate input** property, but promote it. The runtime administrator could then enable message validation under certain circumstances.

Logging different parts of a message

When you design your service flows, it might be a business requirement to have an audit trail of service requests. You can log all messages that are sent by a service requester using the Message Logger mediation primitive.

The integration developer sets the Message Logger **Root** property to indicate what part of the message should be logged. If the integration developer promotes the **Root** property, the runtime administrator can change its value.

For example, if the integration developer sets the alias for the **Root** property to **LogLevel** and its value to `/body`, then the mediation flow would log the body of a message. At run time, the administrator might want to log the entire message, in which case he could set the value of **LogLevel** to `/`. For more information, see: Message Logger mediation primitive

Connectivity scenarios

You can add value to your business by connecting to existing enterprise information systems (EIS). This section shows some of the ways in which you can reuse your EIS systems.

Description

Many enterprises are looking at ways to make use of existing enterprise information systems as services.

EIS systems handle company-wide information; they let enterprises coordinate their business processes and, generally, give a high quality of service.

Integrating EIS systems is a key feature of both WebSphere Enterprise Service Bus (WebSphere ESB) and WebSphere Process Server. You can integrate existing enterprise applications and databases using various mechanisms, including IBM Adapters and messaging bindings.

You can get new value out of EIS systems by reusing existing, and well-tested, applications and databases to support your changing business requirements. Reuse, through integration, can help you develop services quicker and reduce costs. In addition, you can build on top of systems that you already know to be robust.

Business challenge

Existing EIS systems need to be accessed, rather than significantly changed, in order to generate new value for the business. Newly acquired EIS systems need to be integrated into a larger, or different, business model.

Solution

The connectivity scenarios begin when an IT architect decides that a business has data, or business processes, that can valuably be shared, and that sharing can be achieved through application integration.

The development team uses IBM Integration Designer to develop applications that integrate various EIS systems. IBM Integration Designer helps the developers integrate messaging systems. It also helps them generate services from EIS systems, through the use of adapters together with a wizard. The wizard discovers applications and data on an EIS system, and lets the developers generate services from the discovered applications and data. WebSphere ESB or WebSphere Business Process Manager is used to test, deploy, and administer the new applications.

Implementation highlights

These are the key steps in any implementation:

- Decide which EIS systems to integrate.
- Investigate how to connect to the EIS systems.
- Use IBM Integration Designer to create applications that connect to the EIS systems.
- Use WebSphere Enterprise Service Bus or IBM Business Process Manager to deploy, administer, and run the applications. WebSphere ESB is designed to provide connectivity and mediation solutions. WebSphere Process Server contains the WebSphere ESB functionality and, in addition, is designed to handle business processes.

Products

These are the key products in any implementation:

- IBM Integration Designer, Version 7.5
- WebSphere Enterprise Service Bus, Version 7.5 or IBM Business Process Manager, Version 7.5
- WebSphere Adapters

Note: For simplicity, most of the enterprise application integration scenarios show how you can make existing systems into *service providers*. In addition, most of the scenarios assume that the connection between the service requester and the enterprise service bus is made through a Web Service using SOAP/HTTP.

Overview of imports and exports

In order to connect to diverse systems, you must define imports and exports. This section explains the concepts.

Introduction

An export allows WebSphere Enterprise Service Bus (ESB) or WebSphere Business Process Manager to expose a service, so that the service can be called by a service requester. An import allows WebSphere ESB or WebSphere Process Server to call a service.

You can use WebSphere Integration Developer to create service component architecture (SCA) modules that you deploy to WebSphere ESB or WebSphere Process Server. You define which exports the SCA modules have (that is, how service providers can call the SCA modules) and which imports the SCA modules have (that is, which service providers the SCA modules can call). Both imports and exports have *bindings* that define the communication mechanism; for example, Web services bindings (SOAP/HTTP or SOAP/JMS).

Both WebSphere ESB and WebSphere Business Process Manager support a special type of SCA module, called a *mediation module*, that helps you connect to diverse systems.

Figure 59. Simplified example of a message flowing through an ESB

Note:

- SCA terminology calls import bindings and export bindings: *bindings*. However, *bindings* sometimes need *data bindings* and *header bindings*. For example, if you use a SOAP/HTTP binding, the run time knows the exact structure of the messages that it processes; however, when you use other bindings, you might need a data binding or a data handler to clarify the structure of the message. There are some pre-supplied data bindings and data handlers, or you can write your own.
- Web service bindings support either SOAP/HTTP or SOAP/JMS; they do not support SOAP/MQ or SOAP/MQ-JMS. A Web service binding that uses SOAP/JMS supports the JMS point-to-point mode; it does not support the JMS broadcast mode.

Supported bindings

WebSphere Enterprise Service Bus and WebSphere Business Process Manager support the following export bindings:

Figure 60. Supported export bindings

WebSphere Enterprise Service Bus and WebSphere Business Process Manager support the following import bindings:

Figure 61. Supported import bindings

If you want to integrate enterprise information systems (EIS), you generally use messaging bindings (JMS, generic JMS, MQ and MQ JMS) or EIS bindings. EIS exports and imports are created with a particular resource adapter and let you interact with many EIS systems.

Overview of adapters

In order to connect to enterprise information systems (EIS) you might need to use adapters. This section introduces adapter concepts and usage.

Introduction

IBM adapters allow you to integrate enterprise applications, and data storage facilities, in a service-oriented way. Adapters expose low-level EIS functions, or events, in the form of a service.

Adapters are sometimes referred to as resource adapters, and provide a standard interface to proprietary systems. Using standard interfaces avoids the maintenance issues associated with non-standard solutions.

Some adapters come with IBM Integration Designer and are licensed for production use, others are only for development purposes: that is, they can be used to develop and test an application. Generally, after you deploy your application to WebSphere ESB, or WebSphere Business Process Manager, you will need a licensed runtime resource adapter.

Types of adapters

There are two types of IBM adapters:

- WebSphere Adapters, also referred to as JCA adapters.
  - WebSphere Adapters are based on Java EE Connector architecture (JCA), and are the recommended adapters to use with WebSphere ESB and WebSphere Business Process Manager.
  - If an application is developed with a WebSphere Adapter embedded, the adapter is deployed with the application.
- WebSphere Business Integration (WBI) Adapters.
  - WebSphere Business Integration Adapters reside outside of WebSphere ESB or WebSphere Business Process Manager. The run time communicates with this type of adapter through a Java Message Service (JMS) transport layer.

Both types of adapter can be split into two classes:

- Technology Adapters.
  - Technology adapters let you integrate files, FTP, databases and e-mail: these adapters come with IBM Integration Designer and are licensed for production use with WebSphere ESB and WebSphere Business Process Manager.
- Application Adapters.
  - Application adapters let you integrate enterprise application suites, such as SAP Exchange Infrastructure (XI).

External service wizard

Resource adapters, in IBM Integration Designer, are used in the context of an import or an export. You develop an import or an export with the external service wizard and, in developing it, include the resource adapter.

EIS exports and imports are created with a particular resource adapter, and let you interact with EIS systems. The external service wizard follows a similar pattern, regardless of the adapter you are using.

How to connect to WebSphere MQ

You can connect to WebSphere MQ in a number of ways. This section shows some of the options.

Introduction

The following scenarios can be used to communicate with either WebSphere MQ or with WebSphere Message Broker, because WebSphere Message Broker can exchange messages with WebSphere MQ. Therefore, you could connect to WebSphere MQ, and from WebSphere MQ connect to WebSphere Message Broker.

Connect using native MQ

If you want to communicate with native WebSphere MQ applications, you can use MQ bindings. MQ bindings communicate with WebSphere MQ using native MQ messages (the Message Queue Interface MQI).

The MQ bindings need *data bindings*, and there are a number of MQ data bindings supplied.

Note:

- You can only use MQ bindings with "remote" queue managers, through a WebSphere MQ client connection; you cannot use WebSphere MQ bindings with local queue managers. However, you can always connect to a "local" queue manager, (one on the same machine as the WebSphere ESB or WebSphere Business Process Manager), as if it were remote by connecting through the client connection.
- MQ bindings only support point-to-point messaging (although WebSphere MQ itself supports publish/subscribe messaging).

Figure 62. Using an MQ binding

Connect using MQ JMS

If you want to communicate with Java Message Service (JMS) applications deployed to WebSphere MQ, (applications using WebSphere MQ as the JMS provider), you can use MQ JMS bindings. MQ JMS bindings let you communicate with WebSphere MQ using JMS messages (the JMS API).

The MQ JMS bindings need *data bindings*, and there are a number of MQ JMS data bindings supplied.

Note: MQ JMS bindings support point-to-point messaging and publish/subscribe messaging (publish/subscribe messaging is for one-way operations only).

Figure 63. Using an MQ JMS binding

How to connect to databases

You can connect to numerous databases, and in a variety of ways. This section shows some of the options.

Introduction

You might have databases that you want to use as service providers. Alternatively, you might want to use a database to trigger a service request, or use a database to add details to a service request, before the request is sent to the service provider.

You can use adapters to access databases as service endpoints, and you can use mediation modules to update service requests with information from databases.

The WebSphere Adapter for JDBC

Generally, if you want to use a database as a service requester, or a service provider, you use an adapter.

The WebSphere Adapter for JDBC is a JCA adapter that can be used with any database that has a JDBC driver supporting the JDBC 2.0 specification, or a later specification. For example: DB2®, Oracle, Microsoft SQL Server, and Sybase. The JDBC adapter supports inbound and outbound processing.

EIS imports and exports let you connect from WebSphere ESB, or WebSphere Business Process Manager, to resource adapters such as the WebSphere Adapter for JDBC. With an EIS import, a service requester can access a database or call a program on an enterprise information system (EIS) system. With an EIS export, an EIS application can call a service provider on another system.

Connect using a JCA adapter: databases as service providers

You can connect to any database that has a JDBC driver supporting the JDBC 2.0 specification, using the WebSphere Adapter for JDBC. WebSphere ESB and WebSphere Process Server can convert data between the business objects that the JDBC adapter outputs, and the interface that the service requester expects.

This solution gives you flexibility and scalability: the JDBC adapter can be replaced with a different type of adapter or binding if the service provider changes.

Figure 64. Using a WebSphere JDBC adapter

Connect using a JCA adapter: triggering service requests from database events

You can set up database trigger events and use them to trigger Web services:

1. Events are triggered by records being written to a database table.
2. The JDBC adapter gets the event output and creates business objects. Business objects are containers for application data that represents business elements, such as a database table or the result of an SQL query. The adapter understands the data format provided by the application, and can process the data, perform the operation, and send the results back in that format.
3. WebSphere ESB, or WebSphere Process Server, can convert the business objects to a format that the Web services interface expects.

Figure 65. Using a JDBC adapter

Connect using a mediation module: updating service requests using databases

You can access databases while service requests are flowing through WebSphere ESB or WebSphere Business Process Manager.

For example, you might want to update messages with data from a database, or log part of a message to a database. To do this you can use the mediation primitives that are the building blocks of mediation modules.

- The Database Lookup mediation primitive lets you look up information in a database table, and store the information in the message being processed.
- The Message Logger mediation primitive lets you log a message to a database. The schema used for the table that the message is logged to, is fixed. Typically, database create or update operations that need more flexibility would use a JDBC adapter.
- The Service Invoke mediation primitive lets you call a service endpoint. The service endpoint can be an EIS system, if you use a reference and an EIS import. You can create an EIS import using the IBM Integration Designer external service wizard, and a resource adapter such as the WebSphere Adapter for JDBC.

Figure 66. Using mediation primitives inside a mediation module

How to connect to SAP XI

You can connect to SAP Exchange Infrastructure (SAP XI) in a number of ways. This section shows some of the options.

Connect using a JCA adapter

You can connect to SAP XI using the WebSphere Adapter for SAP Software, which is a JCA adapter.

The WebSphere Adapter for SAP Software connects to SAP systems running on SAP Web application servers. The adapter creates a standard interface to the applications and data on the SAP server, and the interface hides the lower-level details.

The adapter works with the SAP server by sending requests to it and by monitoring and responding to events that occur in the SAP server. The adapter supports the Business Application Programming Interface (BAPI) for outbound processing, Application Link Enabling (ALE) for both outbound and inbound processing, and the SAP Query Interface (SQI) for outbound processing.

WebSphere JCA adapters, such as the SAP adapter, appear in a mediation module as an enterprise information system (EIS) export or import.

Figure 67. Using a JCA adapter

Connect using a Web service

SAP provides a SOAP adapter that can expose a business function as a Web service. Because WebSphere ESB and WebSphere Process Server provide support for Web services connections, you can connect to SAP using an import with a Web services binding.

Both WebSphere ESB and WebSphere Business Process Manager can mediate between the Web service interface used by SAP, and the interface expected by the service requester.

Figure 68. Using Web services

Connect using JMS

You can also connect to SAP XI using JMS. You can use the SAP JMS adapter, which exposes a business function through JMS.

The JMS provider you use must be supported by both the enterprise service bus, (WebSphere ESB or WebSphere Business Process Manager), and the SAP JMS adapter. Therefore, WebSphere MQ is a suitable JMS provider.

The ESB connects to the JMS provider using an appropriate binding on an import. If you are using WebSphere MQ, the binding can be an MQ JMS binding.

Figure 69. Using JMS

How to connect to CICS

You can connect to CICS in a number of ways. This section shows some of the options.

Connect using a JCA adapter

If you use the CICS Transaction Gateway, you can connect to CICS using the IBM CICS ECI Resource Adapter, which is a JCA adapter. The CICS ECI Resource Adapter is shipped with IBM Integration Designer and the CICS Transaction Gateway.

When you use the CICS ECI Resource Adapter, CICS programs can act as service providers (they cannot act as service requesters).

The IBM Integration Designer external service wizard works with the CICS ECI Resource Adapter to create services that call programs on a CICS server. You use the external service wizard to create a CICS import: the wizard helps you to configure the adapter; and also to develop and generate a service. The CICS import is an import with an enterprise information system (EIS) binding that connects to the CICS ECI Resource Adapter, which in turn connects to the CICS Transaction Gateway.

Figure 70. Using a CICS JCA adapter

Connect using a Web service

CICS provides a Web service interface that allows you to integrate CICS programs using Web services. Because WebSphere ESB and WebSphere Process Server provide support for Web services, you can connect to CICS using the CICS Web service interface. At run time, WebSphere ESB or WebSphere Business Process Manager, mediates between the Web service interface provided by CICS, and the interface expected by the service requester or service provider.

The CICS Web service interface is bidirectional; therefore, CICS applications can act as service requestors or service providers. If a CICS program needs to act as a service requester, you must create a Web services export to connect to the CICS Web service; if a CICS program needs to act as a service provider, you must create a Web services import to connect to the CICS Web service.

This scenario involves working with the CICS Web service support; therefore, it requires some knowledge in this area. For more information on implementing CICS Web services, see: Application Development for CICS Web Services.

Figure 71. Using the CICS Web service interface

Connect using the WebSphere MQ-CICS bridge

WebSphere MQ for z/OS has an MQ-CICS bridge, so you can communicate with CICS through WebSphere MQ. WebSphere Enterprise Service Bus and WebSphere

Business Process Manager provide MQ *bindings* on imports and exports, and you can use MQ bindings to communicate either with WebSphere MQ or with the WebSphere MQ-CICS bridge.

If you want to use the MQ-CICS bridge you need to create *data bindings* for the MQ body and header: the data bindings must construct MQ messages in the correct format for the bridge.

When you use IBM Integration Designer to generate a WebSphere MQ import binding, the tools can help you generate an MQ binding with suitable data bindings for the operation. To construct a WebSphere MQ *message body* that can be handled by the bridge, you need to select the data binding option: **MQ adapter language data binding generator**.

Figure 72. Using the MQ-CICS bridge

You can set, clear, or retrieve some information carried in the MQ message header structures defined by the MQ CICS Bridge. For example, you can set values in the `MQCIH` header, then send the MQ message to a CICS application, through the MQ CICS Bridge. In WebSphere Enterprise Service Bus, `MQCIH` header values can be set dynamically using meditation primitives, or statically by setting MQImport binding properties. In WebSphere Process Server, `MQCIH` header values can be set statically in a business module by setting MQImport binding properties.

From your WebSphere Enterprise Service Bus or WebSphere Process Server application, you can create a call to a CICS service, through the MQ CICS bridge. Your application can handle the response from the CICS service. The response could be a fault message. To handle the response, use IBM Integration Designer to add an MQ Import connecting to the WebSphere MQ queue supported by the MQ CICS Bridge. You then configure your application to add the required `MQCIH` header to the message. When your application runs, an MQ request message is sent to the CICS service through the MQ CICS Bridge. The CICS service returns a response to your application through the MQ CICS Bridge. The MQ headers in the response will include `MQCIH` headers. The response could be a fault message, and must be handled.

A WebSphere Enterprise Service Bus application can mediate a request from an MQ Client, where the request already has an `MQCIH` header. This request can be forwarded on to a CICS service, through the MQ CICS Bridge. The WebSphere Enterprise Service Bus application can add, modify, or remove the `MQCIH` header, and returns the CICS response to the client. The response could be a CICS fault, and must be handled correctly.

A WebSphere Enterprise Service Bus application can mediate a request from an MQ Client, where the request already has an `MQCIH` header, and is received from the MQ CICS Bridge. The WebSphere Enterprise Service Bus application can add, modify, or remove the `MQCIH` header, and then forward the request to a non-CICS service. The service response is returned to the client.

How to connect to IMS

You can connect to IMS™ in a number of ways. This section shows some of the options.

Connect using the WebSphere MQ-IMS bridge

WebSphere MQ for z/OS has an MQ-IMS bridge, so you can communicate with IMS through WebSphere MQ. WebSphere Enterprise Service Bus and WebSphere Business Process Manager provide MQ *bindings* on imports and exports, and you can use MQ bindings to communicate either with WebSphere MQ or with the WebSphere MQ-IMS bridge.

If you want to use the MQ-IMS bridge you need to create *data bindings* for the MQ body and header: the data bindings must construct MQ messages in the correct format for the bridge.

When you use IBM Integration Designer to generate a WebSphere MQ import binding, the tools can help you generate an MQ binding with suitable data bindings for the operation. To construct a WebSphere MQ *message body* that can be handled by the bridge, you need to select the data binding option: **MQ adapter language data binding generator**.

You can set, clear, or retrieve some of the header values carried in the MQ message header structures defined by the MQ-IMS Bridge. For example, you can set values in the `MQIIH` header, then send the MQ message to an IMS application, through the MQ-IMS Bridge. The `MQIIH` header values can be set statically, or dynamically, using meditation primitives such as the MQ Header Setter primitive.

From your WebSphere ESB or WebSphere Process Server application, you can create a call to an IMS service, through the MQ-IMS bridge. Your application can handle the response from the IMS service. The response could be a fault message. To call an IMS service from your application, use IBM Integration Designer to add an MQ Import connecting to the WebSphere MQ queue supported by the MQ-IMS Bridge. You then configure your application to add the required `MQIIH` header to the message. When your application runs, an MQ request message is sent to the IMS service through the MQ-IMS Bridge. The IMS service returns a response to your application through the MQ-IMS Bridge. The MQ headers in the response will include `MQIIH` headers. The response could be a fault message, and must be handled.

Your application can mediate a request from an MQ Client, where the request already has an `MQIIH` header. This request can be forwarded on to an IMS service,

through the MQ-IMS Bridge. The WebSphere ESB application can add, modify, or remove the `MQIIH` header, and return the IMS response to the client. The response could be an IMS fault, and must be handled correctly.

Your application can mediate a request from an MQ Client, where the request already has an `MQIIH` header, and is received from the MQ-IMS Bridge. The WebSphere ESB application can add, modify, or remove the `MQIIH` header, and then forward the request to a non-IMS service. The service response is returned to the client.

Service gateway scenarios

A service gateway provides a single access point for multiple services. This section shows some of the ways in which you can use service gateways.

Description

Many enterprises would like to access multiple services through one generic interface. Service gateways act as proxies to services and, therefore, allow you to access multiple services from one address. In addition, service gateways encapsulate transformations, routing, and common processing.

Service gateways are a feature of both WebSphere ESB and WebSphere Process Server. They simplify access to services by providing a common access point.

Figure 73. Multiple access points compared to one access point

There are a number of service gateway patterns that you can use, depending on your business requirements, for example:

- Static service gateway, with or without multiprotocol support
- Dynamic service gateway, with or without multiprotocol support
- Proxy gateway
- Mediation policy service gateway
- Web service standards service gateway

IBM Integration Designer contains a patterns explorer that you can use to develop service gateways and proxies.

Business challenge

The general business requirement is that a set of services needs to be accessed in the same way. In addition, there might be other requirements, for example:

- The service gateway must provide common processing, such as auditing.
- The service gateway must allow services to be dynamically added or removed.
- The service gateway must access legacy services.
- The service gateway must configure requests, on a message-by-message basis.
- The service gateway must provide qualities of service, such as security.

Solution

The service gateway scenarios begin when an IT architect decides that a business has a number of services that need to be accessed as a set, using a single network-addressable endpoint.

Whatever service gateway pattern the architect chooses, the developer must create a module using the predefined Service Gateway interface, supplied with IBM Integration Designer.

In addition to the Service Gateway interface, IBM Integration Designer provides a patterns explorer that creates service gateways according to a number of different patterns. You can access the patterns explorer in the following way: **Window** > **Show View** > **Patterns Explorer** > ***gateway_pattern***. Generally, after using the patterns explorer you need to further develop the module for your own needs. Alternatively, you can create your own module without using the patterns explorer. Depending on the service gateway pattern, you would add whichever mediation primitives produce the required functionality.

- If you are creating a dynamic service gateway, you might need to add the endpoint information to WebSphere Service Registry and Repository (WSRR).
- If you are creating a proxy gateway, you must add the endpoint information to the built-in configuration store.
- If you are creating a mediation policy service gateway, you must add your SCA modules and mediation policies to WSRR. If you are associating mediation policies with target services, you must add the target services to WSRR. You can use Business Space widgets to create your mediation policies in WSRR.

You can use IBM Integration Designer to do some initial testing of your solution. Then you use either WebSphere ESB or WebSphere Process Server to test, deploy, and administer your new module.

Implementation highlights

These are the key steps in any implementation:

- Plan your service gateway: review the different service gateway patterns and decide which best fits your requirements.
- Use IBM Integration Designer to create applications that define your service gateway.
- Optional: If you choose a dynamic service gateway, use WSRR to store any endpoint information.
- Optional: If you choose a mediation policy service gateway, use WSRR to store any SCA modules and target services that you need. Use Business Space widgets to create your mediation policies.

- Optional: If you choose a proxy gateway pattern, use Business Space widgets to add services to proxy groups.
- Use WebSphere Enterprise Service Bus or WebSphere Process Server to deploy, administer, and run the applications.

Products

These are the key products in any implementation:

- WebSphere Integration Developer, Version 7.0
- WebSphere Enterprise Service Bus, Version 7.0 or WebSphere Process Server, Version 7.0
- Optional: WebSphere Service Registry and Repository, Version 7

Patterns for service gateways and proxies

You can develop service gateways and proxies using the patterns explorer in IBM Integration Designer. The purpose of the various patterns supported by the patterns explorer.

Introduction

Service gateways and proxies can vary from the simple to the complex. The IBM Integration Designer patterns explorer helps you develop service gateways and proxies, according to standard patterns.

Simple service proxy: static and dynamic patterns

The simple service proxy pattern is the simplest pattern. You can proxy a single type of service. This pattern creates a module that you can deploy without further work.

The simple service proxy pattern creates a one-to-one mapping between a service requester and a service provider, and is not intended for message transformations.

You can create a static simple service proxy, by specifying the service endpoint at development time. Alternatively, you can create a dynamic simple service proxy, in which case the module retrieves a service endpoint from WSRR. However, even when you use WSRR queries to change service endpoints, the type of service remains the same.

Service gateway: Proxy gateway pattern

The proxy gateway pattern extends the simple service proxy pattern, so you can proxy multiple different services. This pattern creates a module that you can administer using a Business Space widget.

The proxy gateway pattern lets you create a many-to-many mapping between service requesters and service providers. However, one inbound message maps to one service endpoint.

The proxy gateway pattern supports only Web services.

Service gateway: Static service gateway pattern

The static service gateway pattern creates a service gateway that proxies multiple services with known endpoints, when you develop the module. This pattern produces a module that is comparable to having multiple simple service proxies of the static type.

Using the static service gateway pattern can let you create a many-to-many mapping between service requesters and service providers.

After using the static service gateway pattern, you must configure your module. In addition, you might want to add further logic to the mediation flow before deploying the module. For example, you can access or update the body of the message. It is easier to access the message body when using a static service gateway then when using a dynamic service gateway.

Service gateway: Dynamic service gateway pattern

Use the dynamic service gateway if you have multiple services and message formats that can change on a regular basis. The dynamic service gateway requires more planning and administration than the other service gateways or proxies. However, it provides a powerful and flexible way to make use of your services.

The dynamic service gateway pattern creates a service gateway that proxies multiple services, whose endpoints you retrieve at run time. You can retrieve the endpoints from a number of sources, including WSRR.

With the dynamic service gateway pattern you can create a many-to-many mapping between service requesters and service providers.

After using the dynamic service gateway pattern, you must configure your module. In addition, you should add further logic to the mediation flow before deploying the module.

If you require a dynamic service gateway to access or update the body of the message, you must select the box that inflates the message. In addition, you must ensure that the correct schemas are available to the module, for example, in a library. A dynamic service gateway does not select which service endpoint or message format it will use, in advance. Therefore, if you inflate the message body you should choose to use mediation policies to customize your service gateway behavior, depending on the message type you are handling.

Overview of proxy gateways

A proxy gateway is an SCA module that receives Web service requests and forwards the requests to a defined endpoint. The endpoints are defined in groups called proxy groups. This topic explains the concepts, and also explains how the development and administration teams interact to produce proxy gateways.

Introduction

A proxy gateway is an SCA module containing a Gateway Endpoint Lookup mediation primitive that has one or more proxy groups. Each proxy group can contain one or more *virtual service names*. Using Business Space widgets, you can create associations between the virtual service names and existing service endpoints. The proxy gateway can then forward Web service requests to the

service endpoints. The associations are stored in a built-in configuration store, either in WebSphere Enterprise Service Bus (WebSphere ESB) or in WebSphere Process Server.

Details

A proxy gateway module has the following characteristics:

- One inbound message maps to one outbound message.
- All outbound messages have the same qualities of service (Qos).
- Message routing can be done using one of the following methods:
  - URL-based routing.
  - XPath-based routing.

When you develop a proxy gateway module, you configure the Gateway Endpoint Lookup primitive by specifying a point in the request message where a virtual service name can be found. The virtual service name can be found using the URL or an XPath. You can also define the name of the proxy group where the virtual service name exists, or take the default name.

After you create your proxy gateway module, you install the module on either WebSphere ESB or WebSphere Process Server. You can then use Business Space widgets to associate your virtual service names with real service endpoints. When you save the association, the details are stored in the built-in configuration store. A virtual service name can have a list of endpoints associated with it, in which case you determine the order in which the services are tried.

To create a Web service client you need the WSDL that contains the proxy gateway endpoint and the virtual service name.

Note: The service endpoints that you provide to the Business Space widgets must be accessible, both when you define the associations and when you retrieve the WSDL for the client to use. The service endpoints must be accessible because the proxy gateway uses them to retrieve the most recent WSDL.

Figure 74. Overview of a proxy gateway request

Overview of development and administrative tasks

To create a proxy gateway, the developer and the runtime administrator must work together. They must create and integrate the proxy gateway module and the service definitions.

1. From WebSphere Integration Developer:
   a. Use the Patterns Explorer to create a proxy gateway module, which is an SCA module containing a Gateway Endpoint Lookup mediation primitive. **Window** > **Show View** > **Patterns Explorer** > **Proxy Gateway**. Specify the lookup method (URL or XPath), and the proxy group or groups.
   b. Promote the module properties that you want to override, assigning each property an appropriate alias and group.
   c. Optional: If you want WS-Policy information, you can specify the information on the module export and the module import.
   d. Export the SCA module.

      Note: If you have created a versioned SCA module, you must export your module as a command-line deployment. Otherwise, you can export your SCA module in an EAR file.
2. From WebSphere ESB or WebSphere Process Server:
   a. Optional: If you have a versioned SCA module, you must run **`serviceDeploy`** on each of the exported modules to generate an EAR file.
   b. Optional: If you have a versioned SCA module that you want to deploy to clusters, you must run **`createVersionedSCAModule`**. The command creates a new EAR file that contains a unique instance of a versioned SCA module. You must create one instance of the module for every cluster you want to deploy to.

      Note: You must load your versioned SCA modules into WSRR from the new EAR files.

c. Install your proxy gateway module.
d. In Business Space, use the Proxy Gateway widget to add virtual service names and service endpoints to your proxy groups.

   Note: The endpoint addresses that you provide to the Business Space widgets must refer to accessible endpoints.
e. To retrieve the WSDL needed to create a client, submit a request to `http://gateway.com/virtualservicename?wsdl` where *http://gateway.com* is the URL of the proxy gateway export, and *virtualservicename* is the name of a particular virtual service. If you defined WS-Policy information on the module export, the WSDL contains the WS-Policy information.

   Note: The service endpoints associated with the virtual service name must be accessible when you submit the request.
f. Use the endpoint address contained in the WSDL to call the proxy gateway. The proxy gateway uses the virtual service name, embedded in the endpoint address, to route your request to the appropriate service. If you are using URL-based routing, the Web service URL is used to determine the endpoint address. If you use XPath-based routing, the endpoint address must exist in the expected place in the message. For further information, refer to the Gateway Endpoint Lookup documentation.

Planning

Before you develop your proxy gateway, you should plan your overall system:

- Decide whether to route based on a URL or an XPath.
- Decide whether you want to have one proxy group, which is the default, or more than one proxy group. You might have more than one proxy group if you wanted to define groups for different categories of user.
- Decide what mediation primitive properties you want to promote.

Mediation policy scenarios

You can control message flows between services by using mediation policies: different mediation policies can apply in different contexts. This section shows some of the ways in which you can use mediation policies to control message flows.

Description

Many enterprises would like to dynamically control their service interactions, by using context information.

You can develop new service interactions that achieve greater levels of flexibility and administrative control, using mediation policies. In addition, you can get new value out of existing systems by adjusting message flows according to the context in which they occur.

Mediation policies are a feature of both WebSphere Enterprise Service Bus (WebSphere ESB) and IBM Business Process Manager that allow you to improve the contextual control of service interactions.

Figure 75. Messages can be altered according to the context in which they occur

Business challenge

Existing services need to be accessed in constantly changing ways. You need to be able to govern service interactions, so that when circumstances change you can adjust the interactions rather than change the applications.

Solution

The mediation policy scenarios begin when an IT architect decides that a business has services or data that needs to be accessed more flexibly, and governed more centrally.

The integration developer uses IBM Integration Designer to develop applications that define and access mediation policies. The WSRR administrator loads the SCA modules and, if required, WSDL documents into WebSphere Service Registry and Repository (WSRR). The WSRR administrator can create mediation policies using WSRR, directly. Alternatively, the runtime administrator can create mediation policies, in WSRR, using Business Space widgets. The runtime administrator uses WebSphere ESB or IBM Business Process Manager to test, deploy, and administer the new applications.

Implementation highlights

These are the key steps in any implementation:

- Plan your mediation policies: this includes deciding how many mediation policies you need and what properties they should contain.
- Use IBM Integration Designer to create applications that define and access mediation policies.
- Use WSRR, to store any SCA modules, WSDL (describing target services), and mediation policies that the run time needs.
- Optional: Use Business Space powered by WebSphere to create mediation policies in WSRR. Both WebSphere Enterprise Service Bus and IBM Business Process Manager include Business Space widgets that help you create mediation policies.

- Use WebSphere ESB or IBM Business Process Manager to deploy, administer, and run the applications.

Products

These are the key products in any implementation:

- IBM Integration Designer, Version 7.5
- WebSphere Enterprise Service Bus, Version 7.5 or IBM Business Process Manager, Version 7.5 (these products include Business Space widgets for creating mediation policies)
- WebSphere Service Registry and Repository, Version 7.5

Overview of mediation policies

To use mediation policies effectively, you should understand the concepts behind them and plan your overall system. This topic explains the concepts, and also explains how the development and administration teams interact to produce mediation policies.

Introduction

When you use IBM Integration Designer to create SCA modules that contain mediation flows, any module property that you choose to promote (make visible to the run time) is also a dynamic property, so long as the property exists at the top level. Dynamic properties can be overridden, at run time, using mediation policies in a registry. After you deploy an SCA module to your run time, you can see the dynamic module properties, and values, from the administrative console; using mediation policies you can override these values. Whether property values are overridden for a service interaction depends on the message being processed and the configuration of the mediation policy.

Details

When you export your SCA module, IBM Integration Designer generates a default mediation policy for each property group in your SCA module. For example, if your SCA module contains three property groups,IBM Integration Designer generates three default mediation policies each containing all the dynamic properties belonging to one group. The default mediation policies represent the values given to all dynamic properties, at development time. Although IBM Integration Designer generates default mediation policies, it does not attach them to the SCA module. You must decide how to use the default mediation policies.

Generally, you should use default mediation policies without any associated conditions, and create your own mediation policies if you want to have conditional mediation policies. Before you can create conditional mediation policies, in WebSphere Service Registry and Repository (WSRR), you must define the conditions in IBM Integration Designer In WSRR, you create conditions on the mediation policy attachment that WSRR generates when you attach a mediation policy to an SCA module or a target service. At run time, the conditions (sometimes referred to as gate conditions) must be met before the mediation policy can be used.

Note: Mediation policies with gate conditions have a higher precedence than mediation policies without gate conditions. For more information, see the: the mediation policy processing model.

You can use WSRR classifications with your mediation policies. WSSR supports many classification systems, including life cycle classifications that you can use to govern your mediation policies.

Overview of development and administrative tasks

In order to use mediation policies, the developer, the runtime administrator, and the WSRR administrator must work together. They must create and integrate the SCA modules, target services and mediation policies required. You can either create mediation policies using WSRR, directly, or you can use Business Space widgets to create mediation policies in WSRR. This overview of development and administrative tasks assumes that you are not using Business Space widgets. For more information on using Business Space widgets, see the mediation policy tutorials and the administrative tasks.

1. From IBM Integration Designer:
   a. Create an SCA module containing a Policy Resolution mediation primitive. Generally, any other mediation primitives in the SCA module should come after the Policy Resolution mediation primitive, and have at least one dynamic property.
   b. Optional: If you want to associate mediation policies with a target service rather than a module, you should add an Endpoint Lookup primitive before the Policy Resolution primitive.
   c. Promote the module properties you want to override, assigning each property an appropriate alias and group.
   d. Optional: Create conditions on the Policy Resolution mediation primitive.
   e. Optional: Create classifications on the Policy Resolution mediation primitive. The classifications must match the WSRR classification URIs.
   f. Export the SCA module.

      Note: If you have created a versioned SCA module, you must export your module as a command line deployment; otherwise, you can export your SCA module in an EAR file.
2. From the run time:
   a. Ensure you have a suitable WSRR definition for your SCA module.
   b. Optional: If you have a versioned SCA module, you must run **`serviceDeploy`** on each of the exported modules to generate an EAR file.
   c. Optional: If you have a versioned SCA module that you want to deploy to clusters, you must run **`createVersionedSCAModule`** to create a new EAR file that contains a unique instance of a versioned SCA module. You must create one instance of the module for every cluster you want to deploy to.

      Note: You must load your versioned SCA modules into WSRR from the new EAR files.
   d. Deploy the SCA module.
3. From WSRR:
   a. Load the SCA module.
   b. Optional: Load the WSDL for the target service you want to use.
   c. Optional: Create your own mediation policies.
   d. Attach the appropriate mediation policies either to your SCA module or to your target service, or both.
   e. Optional: Create gate conditions.

f. Optional: Make your policy document governed.
g. Optional: Add non-life-cycle classifications to your policies.

Planning

Before you develop mediation policies you should plan your overall system:

- Decide which WSRR instance you will use.
- Decide what mediation primitive properties need to be overridden using mediation policies.
- Decide on the property group name (or names). The group name helps to form the namespace of the mediation policy.
- Decide if you want to associate mediation policies with your SCA module or your target service, or both.
- Decide on your mediation policy pattern.
- Optional: Decide if you need to create your own mediation policies, rather than use only the default mediation policies. If you choose to create your own mediation policies, you should decide on their identifiers. The namespace and identifier, when taken together, must be unique across the instance of WSRR that contains the mediation policy.
- Decide if you need any conditional mediation policies. At run time, gate conditions must be satisfied before a conditional mediation policy can be used. In IBM Integration Designer, every gate condition must be specified on the Policy Resolution mediation primitive as a **Policy condition name**. In WSRR, the gate conditions are specified on the mediation policy attachment.
- Decide if you want to use WSRR classifications. You must specify the classifications you want to use, in both IBM Integration Designer and in WSRR.

Example: Mediation policies associated with modules

This example shows a ticket pricing system where mediation policies are associated with the module, not the target services.

Overview

The following example shows an SCA module, called applyDiscountPolicy, that receives messages containing the age of a ticket buyer. Mediation policies are used to retrieve the ticket discount that should be applied.

The design is to have one default mediation policy and two conditional mediation policies (the condition being the age of the ticket buyer). Mediation flows will use the conditional mediation policies when particular conditions apply, and the default mediation policy when no specific conditions apply.

Note: Mediation policies are not designed for creating complex business processes. Typically, if you require complex business processing you should consider the business process support found in WebSphere Process Server.

To achieve this design, the integration developer creates an SCA module that has one mediation flow component and one property group. The mediation flow component contains one Policy Resolution mediation primitive, which has one dynamic (promoted) property: **discount**, and one condition (**Policy condition name**): **age**. The condition has an XPath of: `/body/input/age` (which tells the run time whereabouts in the message the condition can be found). In addition, the

mediation flow component contains two other mediation primitives: one to do a database query and one to process the output from the database query.

When the SCA module is exported, WebSphere Integration Developer creates one default mediation policy (because there is only one property group).

When the WSRR administrator loads the SCA module, the SCA module is loaded together with the default mediation policy. The WSRR administrator, or the Business Space administrator, then creates two new mediation policies, both of which contain the **discount** property. The WSRR administrator, or the Business Space administrator, attaches the default mediation policy and the two user-defined mediation policies to the SCA module. In addition, the administrator creates conditions for the user-defined mediation policies. One mediation policy can only be used for people over the age of 59, and the other mediation policy can only be used for people under the age of 12.

What happens at run time

At run time, the SCA module receives a message and queries the WSRR instance that has been loaded with the applyDiscountPolicy module, and its default mediation policy.

WSRR returns whichever mediation policy fits the message being processed. The mediation policy provides the mediation flow with the discount that should be applied: 10 percent, 5 percent or 0 percent. After the run time knows the discount, the applyDiscountPolicy module can call a database to get the normal ticket price. Lastly, the normal ticket price and the discount percentage are passed to a Custom Mediation primitive, which calculates the ticket price. After the applyDiscountPolicy module has completed its processing, it passes the new message (containing the ticket price) to the service provider.

Figure 76. At run time, the service provider receives a context-specific message

Example: Mediation policies associated with target services

This example shows an internet education system where the mediation policies are associated with the target services, not the module.

Overview

The following example shows an SCA module, called onlineEducation, that receives messages containing education requests. Currently, the education requests are either for a list of relevant courses, or to buy and download a course. A request for a list of relevant courses is sent to service provider one, and a request to buy and download a course is sent to service provider two. Each service has different audit requirements: service two needing to log requests and service one not needing to log requests. Therefore, different mediation policies are associated with the two services.

To achieve this design, the integration developer creates an SCA module that has one mediation flow component and one property group. The mediation flow component contains the following mediation primitives:

1. An Endpoint Lookup mediation primitive to retrieve the endpoint for the target service.
2. A Policy Resolution mediation primitive to retrieve the mediation policy associated with the target service.
3. A Message Logger mediation primitive to log the request, if the mediation policy sets the **Enabled** property to true.

When the SCA module is complete, it can be installed on either WebSphere Enterprise Service Bus (WebSphere ESB) or WebSphere Process Server.

Details

The integration developer must know how to configure the Endpoint Lookup mediation primitive, so that the primitive retrieves the correct endpoint from the registry. For example, the developer could use a field in the message to identify the endpoint. To do this you define an XPath expression on the Endpoint Lookup primitive and, in WSRR, create a user property on the appropriate WSDL Port object.

The integration developer promotes the Message Logger property called **Enabled**, and sets its value to false. When the **Enabled** property is false no logging is done.

When the SCA module is exported, IBM Integration Designer creates one default mediation policy (because there is only one property group). The default mediation policy sets the **Enabled** property to false.

When the WSRR administrator loads the SCA module, the default mediation policy is loaded with the SCA module. The WSRR administrator also loads the WSDL documents that define the service providers.

The WSRR administrator, or the Business Space administrator, then creates a new mediation policy that sets the **Enabled** property to true. The WSRR administrator, or the Business Space administrator, attaches the default mediation policy to the WSDL for service provider one and the user-defined mediation policy to the WSDL for service provider two.

Note:

- The Web Services Policy Framework (WS-Policy) specifies that, for every target service, policies can exist at different levels. The WS-Policy calls these levels the policy subjects. The policy subjects are: service, endpoint, operation, and message. In this example the mediation policies are attached to the service subject.
- In this example, there is no need to attach the default mediation policy because the **Enabled** property is already false. However, the module might be expanded and the default policy might then include other properties, either for the Message Logger mediation primitive or for another mediation primitive.

What happens at run time

At run time, the SCA module receives a message and queries the WSRR instance that has been loaded with the getCourse module, the WSDL documents and their mediation policies.

WSRR returns the mediation policy associated with the service being called. The mediation policy determines whether logging is required for audit purposes.

Figure 77. At run time, logging is done when a course is paid for

Service level agreement scenarios

An SLA is a negotiated agreement between a service requester and a service provider, that can specify levels of functionality, performance, and availability. This section shows some of the ways in which you can use SLAs.

Description

An SLA can be a formal or informal contract, and can define items such as service priorities and guarantees.

Many enterprises would like to manage their services by exploiting SLAs. For example, if a service can only process so many messages per second, you could agree a certain number of messages with specified consumers.

Both WebSphere Enterprise Service Bus (WebSphere ESB) and WebSphere Process Server are able to check whether or not a particular service requester has an service level agreement with a particular service provider, using the SLA Check mediation primitive. The SLA Check mediation primitive uses WebSphere Service Registry and Repository (WSRR) to check for a suitable SLA.

WSRR stores SLAs as part of its governance enablement profile. A consumer service representation in WSRR can have an SLA that describes the contract to a provider service definition. This SLA can be checked by WebSphere ESB or WebSphere Process Server, to determine whether a consumer is entitled to invoke a service. Additional metadata stored in the SLA describes the agreed parameters in which the consuming service is allowed to operate. For example, the level of service, minimum and maximum messages per day, and so on.

The following diagram shows two service requests, one made from the Web and one made by an internal user. The external request, made from the Web, has an associated SLA and is therefore it is routed to the production-level service.

Figure 78. Services can be routed to different endpoints, depending on whether the requester has an associated SLA

Business challenge

SLAs help you to manage finite resources. The business challenge is to ensure that a finite resource, such as a service, is used most effectively, by ensuring that the service consumers with the highest priority get the most resources.

Solution

The SLA scenarios begin when an IT architect decides that a business has services that have quality of service characteristics that need to be managed on a per-client basis.

The integration developer uses IBM Integration Designer to develop applications that check for the existence of an SLA between the service consumer and the

service provider. The WSRR administrator defines the SLAs.

Implementation highlights

These are the key steps in any implementation:

- Plan your SLAs: this includes deciding which service consumers and providers need SLAs, and what properties need to be agreed.
- Use IBM Integration Designer to create applications that access SLAs.
- Use WSRR, to define and store the SLAs.
- Use WebSphere ESB or WebSphere Process Server to deploy, administer, and run the applications.

Products

These are the key products in any implementation:

- IBM Integration Designer, Version 7.5
- WebSphere Enterprise Service Bus, Version 7.5 or IBM Business Process Manager, Version 7.5
- WebSphere Service Registry and Repository, Version 7.5

Invocation check based on an SLA between a client and a service provider

This example shows a system where a service level agreement (SLA) is created between a service consumer (client) and a service provider.

Overview

In this scenario the SLA is associated with a number of available endpoints. The mediation must decide whether to proceed with the invocation based on whether there is an active SLA between the two parties that references the required endpoint. The mediation primitive which has this functionality is the SLA Check mediation primitive. You can configure the SLA Check mediation primitive type as follows:

- Set the Registry Name property to the WSRR instance that contains the SLA.
- Set the Endpoint property to the default value, which is an XPath expression: `/headers/SMOHeader/Target/address`.

  Note: You must add one of the WSRR-based endpoint lookup mediation primitives before the SLA Check mediation primitive.
  The XPath expression indicates the location in the message that contains the endpoint. An Endpoint Lookup mediation primitive sets the address in the default location.
- Set the Consumer ID property to an XPath expression, because many clients can call the same mediation. The XPath expression indicates the location in the message that contains the consumer identifier.
- Set the Context ID property to an XPath expression, which indicates which SLA for the particular consumer is being referenced. The XPath expression indicates the location in the message that contains the context identifier.

In WSRR, when using the governance enablement profile to govern your SOA environment, you must create as part of your governance processes:

- a service version

- a service level definition (SLD) and
- an SLA

When you create these objects you specify the endpoints associated with a particular SLD, a consumer identifier associated with a service version, and a context identifier associated with an SLA. The endpoints referenced from an SLD can be refined further using classifications. The SLA Check mediation primitive uses all of these values to decide if the invocation to the selected endpoint can proceed. For more information, see the documentation on the SLA Check mediation primitive in the Information Center.

The following diagram shows how the endpoint, consumer identifier, and context identifier are related to each other, and how they are related to the client and mediation.

Figure 79. How the SLA Check properties relate to the client and the final service provider.

What happens at run time

At run time, the SCA module receives a message and queries the WSRR instance specified in the SLA Check mediation primitive.

If WSRR finds a matching SLA, the **accept** terminal is fired, otherwise, the **reject** terminal is fired.

Related reference:

SLA Check mediation primitive
Use the SLA Check mediation primitive to check that a service level agreement exists.

Endpoint selection with default configuration

This example shows a system where a service level agreement (SLA) is created between a service consumer (client) and a service provider with default configuration.

In this scenario the SLA is associated with a number of available endpoints so the mediation must select between them based on various selection criteria. These selection criteria are the default criteria supplied as part of WSRR and WebSphere Enterprise Service Bus (WebSphere ESB).

Overview

In this scenario the service consumer is the client calling the mediation, and the client has an SLA with the final service provider. An SLA can be associated with a number of suitable endpoints and a particular client can have a number of active SLAs for different providers. Using the SLA Endpoint Lookup mediation primitive you can select a suitable endpoint type from these relationships. You can configure the primitive as follows:

- Set the Registry Name property to the WSRR instance that contains the SLA.
- Set the Consumer ID property to an XPath expression, because many clients can call the same mediation primitive. The XPath expression indicates the location in the message that contains the consumer identifier.
- Set the Context ID property to an XPath expression that indicates which SLA for the particular consumer is being referenced. The XPath expression indicates the location in the message that contains the context identifier.
- Set the Endpoint Classification property to further refine the selection criteria for the endpoint. For example, this particular interaction might only accept endpoints which are in production.

In WSRR, when using the governance enablement profile to govern your SOA environment, you must create as part of your governance processes:

- a service version
- a service level definition (SLD) and
- an SLA

When you create these objects you specify the endpoints associated with a particular SLD, a consumer identifier associated with a service version, and a context identifier associated with an SLA. The endpoints referenced from an SLD can be refined further using classifications. The SLA Endpoint Lookup mediation primitive uses all of these values to select an endpoint. If a suitable endpoint is found, the mediation can dynamically route to the selected endpoint. For more information on the use of the governance enablement profile, see the WSRR Information Center.

The following diagram shows how the endpoint, consumer identifier, and context identifier are related to each other, and how they are related to the client and mediation.

Figure 80. How the SLA Endpoint Lookup properties relate to the client and the final service provider

This diagram shows a client calling a service provider which it has a service level agreement for the interaction. The Service Version is a representation of the client in the registry; a consumer of the provider's service. The SLD is a representation of the provided service in the registry. The SLD contains non-functional characteristics, such as performance metrics, as well as information about the functional endpoint itself. The SLA is between the consumer, as represented by the Service Version, and the full characteristics of the provided service, as represented by the SLD.

What happens at run time

At run time, the SCA module receives a message and queries the WSRR instance specified in the SLA Endpoint Lookup mediation primitive.

If WSRR finds an endpoint which matches the selection criteria specified, the **out** terminal is fired; otherwise, the **noMatch** terminal is fired. If a match is found, the target address for the selected endpoint is updated in the Service Message Object so that you can route dynamically to the particular target.

This scenario is powerful because it requires the SLA to be in an active state for endpoint selection to succeed. Consequently, if there are problems with the particular client and provider interaction, the SLA can be put into an inactive state in WSRR, which blocks all further interactions between the particular provider and client. However, this does not stop valid interactions from other clients using the same provider; allowing consumer-based control.

Related information:

SLA Endpoint Lookup mediation primitive

Tutorial: configure WSRR for use with SLA Endpoint Lookup primitive
This tutorial describes how to configure WSRR for use with the WebSphere ESB SLA Endpoint Lookup mediation primitive.

Endpoint selection with enhanced selection criteria

This example shows a system where a service level agreement (SLA) is created between a service consumer (client) and a service provider with enhanced selection criteria.

In this scenario the SLA is associated with a number of available endpoints. The mediation primitive selects an endpoint based on various criteria. The default selection criteria supplied as part of WSRR and WebSphere Enterprise Service Bus (WebSphere ESB) are further enhanced.

Overview

In this scenario the service consumer is the client calling the mediation, and the client has an SLA with the final service provider. An SLA can be associated with a number of suitable endpoints and a particular client can have a number of active SLAs for different providers. Using the SLA Endpoint Lookup mediation primitive you can select a suitable endpoint type from these relationships. You can configure the primitive as follows:

- Set the Registry Name property to the WSRR instance that contains the SLA.
- Set the Consumer ID property to an XPath expression, because many clients can call the same mediation. The XPath expression indicates the location in the message that contains the consumer identifier.
- Set the Context ID property to an XPath expression that indicates which SLA for the particular consumer is being selected. The XPath expression indicates the location in the message that contains the context identifier.
- Set the Endpoint Classification property to further refine the selection criteria for the endpoint. For example, this particular interaction might only accept endpoints which are in production.

Add in as many extra parameters to the selection criteria as you need by using the **User Parameters** table under the **Advanced** tab. These parameters are not active until the named query installed on WSRR has been modified. The named query the SLA Endpoint Lookup mediation primitive uses is called SLAEndpointLookup and is included in WSRR. This version expects three parameters, matching exactly, and in order:

1. Consumer ID
2. Context ID
3. Endpoint Classification

Extra parameters added using the **User Parameters** table are passed on the query to WSRR but the XPath query which constitutes the SLAEndpointLookup named query must be modified to use these extra parameters. See the module Advanced: Extending the SLA Endpoint Lookup queryin the *Configure WSRR for use with SLA Endpoint Lookup mediation primitive* tutorial. You can further define the selection by using parameters. These parameters can be further classifications of the endpoint; for example, whether it is an external endpoint or not. They can also be any

attributes defined within your WSRR Governance Enablement Profile (GEP). For information on named queries consult the WSRR information center.

In WSRR, when using the GEP to govern your SOA environment, you must create as part of your governance processes:

- a service version
- a service level definition (SLD) and
- an SLA

When you create these objects, you specify the endpoints associated with a particular SLD, a consumer identifier associated with a service version, and a context identifier associated with an SLA. The endpoints referenced from an SLD can be refined further using classifications. The SLA Endpoint Lookup mediation primitive uses all of these values to select an endpoint. If a suitable endpoint is found, the mediation can dynamically route to the selected endpoint. For more information, see SLA Endpoint Lookup mediation primitive.

The following diagram shows how the endpoint, consumer identifier, and context identifier are related to each other, and to the client and mediation.

Figure 81. How the SLA Endpoint Lookup properties relate to the client and the final service provider

The diagram shows a client calling a service provider which it has a service level agreement for the interaction. In this example, the service version represents the client in the registry; a consumer of the provider's service. The SLD is a representation of the provided service in the registry. The SLD contains non-functional characteristics, such as performance metrics, as well as information about the functional endpoint itself. The SLA is between the consumer, as represented by the Service Version, and the full characteristics of the provided service, as represented by the SLD.

What happens at run time

At run time, the SCA module receives a message and queries the WSRR instance specified in the SLA Endpoint Lookup mediation primitive.

If WSRR finds an endpoint which matches the selection criteria specified, the **out** terminal is fired; otherwise, the **noMatch** terminal is fired. If a match is found, the target address for the selected endpoint is updated in the Service Message Object so that you can route dynamically to the particular target.

This scenario is powerful because it requires the SLA to be in an active state for endpoint selection to succeed. If there are problems with the particular client and provider interaction, the SLA can be put into an inactive state in WSRR, which blocks all further interactions between the particular provider and client. However, this does not stop valid interactions from other clients using the same provider; which gives you consumer-based control.

Related information:

SLA Endpoint Lookup mediation primitive

Tutorial: configure WSRR for use with SLA Endpoint Lookup primitive
This tutorial describes how to configure WSRR for use with the WebSphere ESB SLA Endpoint Lookup mediation primitive.

An ESB in a stock quote scenario

The stock quote scenario shows a business scenario for getting stock quotes, that is implemented using the IBM service connectivity scenario.

Introduction

This scenario assumes a financial services company that provides stock quotes. The company has a number of existing information technology (IT) services that must be reused under new business conditions.

A stock quote sample is provided with WebSphere Enterprise Service Bus and WebSphere Process Server. .

Requirements

After analyzing their business processes the company defines the following, SOA-related, business requirements:

- Different levels of service must be provided, depending on the type of customer making the request. At first there will be two types of customers: standard customers and premium customers.
- If the company creates further levels of service, they must be added without affecting existing customers.

The business requirements lead to the following IT requirements:

- Flexibility and reusability requirements. The solution must be flexible, scalable, and reusable; therefore, the services must be loosely coupled.
  - The service requester only needs to know the interface to invoke, rather than the location, or physical nature, of the endpoint.
  - Messages and interfaces must be defined using WSDL.
  - The service requester provides the customer identifier, in the input message; and the ESB routes the service request based on the customer identifier.

– Customer details must be stored in a database and accessed using a customer identifier.
– The solution must be able to transform the format of a request or response, because the service requester and service providers use different message formats.

Stock quote scenario version 1.0

The stock quote scenario uses a customer identifier to look up customer details in a database, and the customer details are used to route the message. The stock quote scenario assumes that an input message needs its message body transforming to make it suitable for a service endpoint.

Figure 82. Service connectivity scenario for stock quotes: overview

In version 1.0 of the scenario, stock quote inquiries are routed to one of two service providers. However, the flexibility and scalability of the ESB mean that you can add service providers, and change the existing service endpoints, without affecting the service requesters.

Version 1.0 of the scenario stores endpoint addresses in a mediation module. However, later versions of the scenario store endpoint addresses in a database, or a registry, for increased flexibility and governance.

The following message flow shows the ESB:

- Logging the input message, (for audit purposes).
- Looking up customer details, using a customer identifier.
- Routing the message, based on the customer details.
- Transforming the message format, so that it is suitable for the service provider.

Figure 83. Service connectivity scenario for stock quotes: version 1.0 message flow overview

Stock quote scenario: flexibility and reuse

How to use adjust the service connectivity logic of the stock quote scenario, to make the solution more flexible.

Introduction

A mediation flow encapsulates service connectivity logic, that is invoked when a message is received by the ESB. A mediation flow helps you to reuse existing services and applications.

Stock quote version 1.0

Initially, the financial services company that provides the stock quotes has a business requirement for two levels of service: `premium` and `standard`. In addition, the business has only two Web services that can provide the relevant business services. Therefore, version 1.0 of the stock quote mediation flow assumes that there will be only two service endpoints, and that the endpoint addresses are known when the mediation flow is created. The flow is dynamic in the sense that the precise service endpoint is selected at runtime, based upon the customer subscription level. However, the flow could be made more dynamic, and more flexible, by removing the service addresses from the mediation flow and storing them elsewhere.

The following image shows version 1.0 of the stock quote scenario. The mediation flow logs the input message for audit purposes, looks up the customer subscription level in a database, routes the message based upon the subscription level, and transforms the message format to make it suitable for the service provider.

Figure 84. Mediation flow for stock quote version 1.0

Stock quote version 1.1: increasing flexibility and reuse

As the financial services company grows it finds that it has a requirement for greater flexibility, and wants to make use of new Web services as they become available, without deploying modules to the runtime. By storing service endpoint addresses in a database, the administrator can manage the use of Web services by making changes to one database, rather than having to redeploy a changed EAR file, perhaps to multiple machines.

The following image shows version 1.1 of the stock quote scenario. The first mediation primitive looks up the customer subscription level in a database and passes the subscription level to the next mediation primitive, which uses the subscription level to look up the service address. The message is then passed on to another mediation primitive which logs it, for audit purposes. Finally, a mediation primitive transforms the message format, to make the message suitable for the service provider. The image shows an import, which is used to provide a default endpoint, in case the registry fails to return a dynamic endpoint.

Figure 85. Mediation flow for stock quote version 1.1: improved flexibility

Stock quote version 1.2: further flexibility and reuse

Over a period of time the financial services company finds it has a business requirement to use services from other companies, so that it can cater for particular geographical or linguistic areas. The financial services company, therefore, decides to store all the service endpoint addresses it uses in a registry. The mediation flow can then extract the service that is most suitable for the customer. This requires adding service endpoint details to a registry, and setting up a query of the registry from the mediation flow, (using the Endpoint Lookup mediation primitive).

The following image shows version 1.2 of the stock quote scenario. The first mediation primitive looks up the customer subscription level in a database and passes the subscription level to the next mediation primitive, which uses the subscription level, and customer location, to look up the most suitable service in a registry. The message is then passed on to another mediation primitive which logs it, for audit purposes. Finally, a mediation primitive transforms the message format, to make the message suitable for the service provider. The image shows an import, which is used to provide a default endpoint, in case the registry fails to return a dynamic endpoint.

Figure 86. Mediation flow for stock quote version 1.2: improved flexibility and reuse

Stock quote scenario: governance

You could adjust the original stock quote scenario to improve the governance of your services.

Introduction

The life cycle of a service within a service-oriented architecture (SOA) requires governance processes. Governance processes ensure that standards and policies are enforced, and that change occurs in a controlled fashion and with the appropriate authority.

You can use the IBM WebSphere Service Registry and Repository (WSRR) with the ESB, to improve governance.

Improving governance of services

The original business requirements of the stock quote scenario did not include life cycling, or versioning, of stock quote services. However, as the financial services company grows, life cycle management might become a high-priority requirement. For example, to comply with audit requirements, or because the company starts to provide services for other companies.

Version 1.2 of the stock quote scenario requires the storing of service endpoints in a registry. When services are stored in a registry, life cycle management becomes easier and more secure.

The service endpoints must be added to the registry, and a custom property created for the use of the stock quote mediation module. The custom property represents the subscription level used by the stock quote mediation module.

In order for the stock quote scenario to make use of an existing registry, you would need to do the following:

1. Add the stock quote service endpoints to the registry.
   - The WSRR administrator publishes the metadata for all the services you might use.
2. Make the objects (service endpoints) governed.
3. Set up your registry to key stock quote URLs against subscription levels. You can then dynamically route customer requests according to subscription levels.
   - The WSRR administrator creates a custom property that represents the subscription level.
4. Configure the stock quote mediation module to fetch metadata from the registry for a service that has a subscription level of a given value.
   - The integration developer adds and configures the Endpoint Lookup mediation primitive, and exports the mediation module within an enterprise archive (EAR) file.
5. Create and configure a registry definition on the run time.
   - The runtime administrator configures a WSRR definition.
6. Deploy the mediation module.
   - The runtime administrator installs the EAR file containing the mediation module.

Using an ESB to route and transform messages

An enterprise service bus (ESB) links together many types of data, applications, and protocols. This section shows you how you could use an ESB to achieve specific goals, such as message protocol conversion, database logging, and dynamic routing.

Before you begin

Understand your business requirements and objectives.

About this task

Use the task if you want to see how the ESB addresses specific goals, such as protocol conversion and dynamic routing. The use cases provide a high-level overview, and link to other topics that provide more details.

Building a service gateway

A service gateway acts as a proxy to a variety of different services. All service requesters interact with a single endpoint address exposed by the gateway. The gateway is responsible for performing a common operation on every message and routing the request to the correct service provider.

Using Integration Designer you can build a gateway service, with which you can route messages to multiple endpoints using a variety of bindings, for example, HTTP, MQ, JMS or Web Services bindings. This allows the gateway to handle different portType elements with a single generic export. Requests containing different portType elements arrive at a generic endpoint, which is the entry into the gateway. These requests can have the headers, body, or both manipulated in a mediation module before being forwarded to the service provider. There are four types of service gateways:

- The *dynamic* service gateway is a generic mediation that can modify the header information in the message. In a dynamic service gateway, the endpoint address is determined at run time.
- The *static* service gateway is a generic mediation that can modify the header and body information in the message. In a static service gateway, the import used by the service gateway has a concrete interface associated.
- The *dynamic body-aware* service gateway is a generic mediation that can modify the header and body information in the message. This patterns differs from the static service gateway in that both the export and import have service gateway interface attached.
- The *multiprotocol* service gateway is a generic mediation that can modify the header and body information in the message. This allows for protocol switching within the service gateway.

  Note: Appropriate WSDL portType definitions have to be available for the static and dynamic body-aware service gateways.

The following bindings are supported in a service gateway scenario:

- Web Services
- JMS
- Generic JMS
- MQ JMS
- MQ
- HTTP

Figure 87. Overview :Actor (service requester) and system (ESB)

WebSphere Integration Developer provides a wizard for service gateway patterns; the wizard accelerates the development of a gateway by providing a template that can be customized. The instructions assume that you are not using the service gateway pattern wizard.

The following topics explain how to use bindings with static, dynamic, dynamic body-aware and multiprotocol service gateways.

Building a static service gateway

The static service gateway is a generic mediation that can modify the header and body information in the message. In a static service gateway, the import used by the service gateway has a concrete interface associated.

About this task

These steps provide an overview. They do not include all instructions for creating a static service gateway. For complete instructions refer to the WebSphere Integration Developer documentation.

Restriction: The static service gateway scenario supports only document literal wrapped and document literal styled Web service messages.

Procedure

1. The integration developer completes the following steps:
   a. Create a new mediation module. Creating mediation modules
   b. Import the predefined Service Gateway interface and related resources into the project. Importing the ServiceGateway interface and schema
   c. Open the mediation module assembly, and add a mediation flow component to the assembly.
   d. Add the Service Gateway interface to the mediation component. Adding interfaces
   e. Add the specific imports for the end services (with associated interfaces) with the correct configuration.
   f. Create an export for the mediation module using the predefined Service Gateway interface.
      - For HTTP bindings, see: Creating exports
      - For JMS bindings, see: Generating an HTTP export binding
      - For Generic JMS bindings, see: Generating a JMS export binding
      - For MQ JMS bindings, see: Generating a generic JMS export binding
      - For MQ bindings, see: Generating an MQ JMS export binding
      - For Web Service bindings, see: Generating Web service bindings for exports
      1) Generate a binding for the export.
      2) Select **Service gateway function selector** for the export.
      3) Select **Native body data handler** for the export.
   g. Generate the mediation flow implementation by completing the following steps: Building mediation flows
      1) Optional: Select a Message Logger mediation primitive.Message Logger mediation primitive.
      2) Select a Message Filter mediation primitive and configure to route the message.Message Filter mediation primitive.

3) Add a Set Message Type mediation primitive to assert the body type to the corresponding native body message, such as TextBody, BytesBody, ObjectBody, MapBody, or StreamBody. Set Message Type mediation primitive.
4) Add a Data Handler mediation primitive and configure it to convert the protocol-based message to the specific business object (BO). Data Handler mediation primitive.
5) Select an XSL Transformation mediation primitive to convert the message type and modify any values required. XSL Transformation mediation primitive.
6) In the response flow, include a Data Handler primitive to convert from a specific BO to a generic native body message, such as TextBody, BytesBody, ObjectBody, MapBody, or StreamBody.Data Handler mediation primitive.
7) Optional: Add a Message Logger mediation primitive to the response flow. Message Logger mediation primitive.

2. The runtime administrator deploys and administers the mediation module by completing the following steps:
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. Installing an SCA module with the console
   b. Start the application associated with the mediation module. Use the runtime administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When the service request invokes the mediation module, the service request is manipulated and routed to the correct endpoint.

Example

Figure 88. Request flow for static service gateway

Figure 89. Response flow for static service gateway

Building a dynamic service gateway

The dynamic service gateway is a generic mediation that can modify the header information in the message. In a dynamic service gateway, the endpoint address is determined at runtime.

About this task

These steps provide an overview. They do not include all instructions for creating a dynamic service gateway. For complete instructions refer to the WebSphere Integration Developer documentation

Procedure

1. The integration developer completes the following steps:
   a. Create a new mediation module. Creating mediation modules
   b. Import the predefined Service Gateway interface. Importing the ServiceGateway interface and schema
   c. Open the mediation module assembly, and add a mediation flow component to the assembly.

d. Add the Service Gateway interface to the mediation flow component. Adding interfaces
e. Create an import for the mediation module using the predefined Service Gateway interface. Generate bindings for the import. The import is configured with the Native body Data Handler.
f. Create an export for the mediation module using the predefined Service Gateway interface.
   - For HTTP bindings, see: Creating exports
   - For JMS bindings, see: Generating an HTTP export binding
   - For Generic JMS bindings, see: Generating a JMS export binding
   - For MQ JMS bindings, see: Generating a generic JMS export binding
   - For MQ bindings, see: Generating an MQ JMS export binding
   - For Web Service bindings, see: Generating Web service bindings for exports

   1) Generate a binding for the export. The export is configured with the Native body Data Handler and Service Gateway Function Selector.
g. Generate the mediation flow implementation by completing the following steps.
   1) Optional: Optional: Log the message to a database.Message Logger mediation primitive
   2) Select a transform primitive to convert the header information.Data Handler mediation primitive
   3) Select a routing mediation primitive (such as the Endpoint Lookup primitive or Database Lookup primitive) that will use information in the service message object (SMO) to determine the final endpoint for the message. This endpoint address should then be populated into the `/header/SMOHeader/TargetAddress` section of the SMO.Mediation primitives.
   4) Add any mediation primitives required in the response flow, for example, logging or transformation primitives.Mediation primitives.

2. The runtime administrator deploys and administers the mediation module by completing the following steps:
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. Installing an SCA module with the console
   b. Start the application associated with the mediation module. Use the runtime administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When the service request invokes the mediation module, the service request is manipulated and routed to the correct endpoint.

Example

Figure 90. Request flow for dynamic service gateway

Figure 91. Response Flow for dynamic gateway binding

Building a dynamic body-aware service gateway

The dynamic body-aware service gateway is a generic mediation that can modify the header and body information in the message. This patterns differs from the static service gateway because both the export and import have the Service Gateway interface.

About this task

These steps provide an overview. They do not include all instructions for creating a dynamic body-aware service gateway. For complete instructions refer to the WebSphere Integration Developer documentation.

The dynamic body-aware service gateway can only work with a subset of protocols supported by WebSphere ESB for z/OS and WSDL message styles, which are:

- Web service: only supported for WSDL message styles DocLit (single and multipart) and DocLit Wrapped.
- MQ, JMS, and HTTP: only supported where the message format is a valid XML document (for example, as created by the UTF8XMLDataHandler), and the WSDL message style is either single part DocLit or DocLit Wrapped.

Procedure

1. The integration developer completes the following steps:
   a. Create a new mediation module. Creating mediation modules
   b. Import the predefined Service Gateway interface and related resources into the project. Importing the ServiceGateway interface and schema
   c. Open the mediation module assembly, and add a mediation flow component to the assembly.
   d. Add the Service Gateway interface to the mediation flow component. Adding interfaces
   e. Create an import for the mediation module using the predefined Service Gateway interface: Generate bindings for the import. The import is configured with the Native body Data Handler.
   f. Create an export for the mediation module using the predefined Service Gateway interface.
      - For HTTP bindings, see: Creating exports
      - For JMS bindings, see: Generating an HTTP export binding
      - For Generic JMS bindings, see: Generating a JMS export binding
      - For MQ JMS bindings, see: Generating a generic JMS export binding
      - For MQ bindings, see: Generating an MQ JMS export binding
      - For Web Service bindings, see: Generating Web service bindings for exports

      1) Generate a binding for the export. The export is configured with the Native body Data Handler and Service Gateway Function Selector.
   g. Generate the mediation flow implementation by completing the following steps.
      1) Check the **Parse the service gateway message** check box on the input node
      2) Select a routing mediation primitive (such as the Endpoint Lookup primitive or Database Lookup primitive) that will use information in the service message object (SMO) to determine the final endpoint for the message. This endpoint address should then be populated into the `/header/SMOHeader/TargetAddress` section of the SMO.Mediation primitives.
      3) Select a policy resolution primitive or create your own primitive that will populate the dynamic property context and configure as required. See.
      4) Select a transform primitive to convert the message transformation.

2. The runtime administrator deploys and administers the mediation module by completing the following steps:
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. Installing an SCA module with the console
   b. Start the application associated with the mediation module. Use the runtime administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When the service request invokes the mediation module, the service request is manipulated and routed to the correct endpoint.

Example

Figure 92. Request flow for dynamic body-aware service gateway

Figure 93. Response flow for dynamic gateway binding

Building a multiprotocol service gateway

The multiprotocol service gateway is a generic mediation that can modify the header and body information in the message. This allows for protocol switching in the service gateway.

About this task

These steps provide an overview. They do not include all instructions for creating a multiprotocol service gateway. For complete instructions refer to the WebSphere Integration Developer documentation.

The multiprotocol service gateway provides the ability to create a cross-protocol service gateway, sometimes referred to as a protocol-switching service gateway. A combination of the multiprotocol pattern with the other service gateway patterns (for static, dynamic, and dynamic body-aware gateways) can be used to provide the level of capability required.

The following protocols are supported:

- Web Services
- MQ
- JMS
- HTTP

It is important to understand that the message encoding format used in the message content is the same across the protocols, or these differences will need to be resolved in the mediation flow component using transformation primitives.

Procedure

1. The integration developer completes the following steps:
   a. Create a new mediation module. Creating mediation modules
   b. Import the predefined Service Gateway interface and related resources into the project. Importing the ServiceGateway interface and schema

c. Open the mediation module assembly, and add a mediation flow component to the assembly.
d. Add the Service Gateway interface to the mediation flow component. Adding interfaces
e. Create an import for the mediation module using the predefined Service Gateway interface: Creating imports Generate bindings for the import. The import is configured with the Native body Data Handler.
f. Create an export for the mediation module using the predefined Service Gateway interface.
   - For HTTP bindings, see: Creating exports
   - For JMS bindings, see: Generating an HTTP export binding
   - For Generic JMS bindings, see: Generating a JMS export binding
   - For MQ JMS bindings, see: Generating a generic JMS export binding
   - For MQ bindings, see: Generating an MQ JMS export binding
   - For Web Service bindings, see: Generating Web service bindings for exports

   1) Generate a binding for the export. The export is configured with the native body data handler and service gateway function selector.
g. Generate the mediation flow implementation by completing the following steps.
   1) Optional: Complete any generic processing on all messages such as logging the message to a database. Message Logger mediation primitive
   2) Select a routing mediation primitive (such as the Endpoint Lookup primitive or Database Lookup primitive) that will use information in the service message object (SMO) to determine the final endpoint for the message. This endpoint address should then be populated into the `/header/SMOHeader/TargetAddress` section of the SMO. In addition the bindingType information should be populated within /headers/SMOHeader/Target/@bindingType attribute.Mediation primitives.
   3) Optional: If protocol headers need to be defined on the outbound request then a MessageFilter should be used to split the requests into their target protocol types.
   4) Optional: Add any header setter primitives required downstream of the MessageFilter
   5) Add any mediation primitives required in the response flow, for example, logging or transformation primitives.Mediation primitives.

2. The runtime administrator deploys and administers the mediation module by completing the following steps:
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. Installing an SCA module with the console
   b. Start the application associated with the mediation module. Use the runtime administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When the service request invokes the mediation module, the service request is manipulated and routed to the correct endpoint.

Example

Figure 94. Multiprotocol service gateway

Dynamic routing

You can route messages in various ways using endpoints defined at integration time or endpoints determined, dynamically, at run time.

Dynamic routing covers message routing where the flow is dynamic but all possible endpoints are predefined in a Service Component Architecture (SCA) module, and message routing where the flow is dynamic and the endpoint selection is also dynamic. In the latter case, the service endpoints are selected from an external source, at run time.

Dynamic endpoint selection

The run time has the capability to route request and response messages to an endpoint address identified by a message header element. This message header element can be updated by mediation primitives, in a mediation flow. The endpoint address could be updated with information from a registry, a database, or

with information from the message itself. Routing of response messages applies only when the response is being sent by a Web service JAX-WS export.

In order for the run time to implement dynamic routing on a request or response, the SCA module must have the `Use dynamic endpoint if set in the message header` property set. Integration developers can set the `Use dynamic endpoint if set in the message header` property or they can promote it (make it visible at run time), so that the runtime administrator can set it. You can view module properties in the Module Properties window. To see the window, click **Applications** > **SCA Modules** > **Module Properties**. The integration developer gives promoted properties alias names, and these are the names displayed on the administrative console.

Registry

You can use IBM WebSphere Service Registry and Repository (WSRR) to store service endpoint information, and then create SCA modules to retrieve endpoints from the WSRR registry.

When you develop SCA modules, you use the Endpoint Lookup mediation primitive to allow a mediation flow to query a WSRR registry for a service endpoint, or a set of service endpoints. If an SCA module retrieves a set of endpoints then it must use another mediation primitive to select the preferred one.

Dynamic routing using registry lookup

Choose which service is used at run time, using WebSphere Service Registry and Repository (WSRR) to lookup service endpoints.

About this task

If your goal is to have service requests routed to endpoints that are selected dynamically, you can achieve this by using a WSRR registry combined with an Endpoint Lookup mediation primitive. The run time supports dynamic endpoint selection, but you must set the `Use dynamic endpoint` property in your mediation module.

If you decide to use service-version-aware routing, service requests are dynamically routed to a specific version of a module.

Figure 95. Actor (service requester) and system (ESB)

To dynamically route services complete the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module or a WSRR registry. You should refer to product documentation for these instructions.

Procedure

1. Optional: If necessary, the WSRR administrator creates and configures a WSRR registry. Installing Websphere Service Registry and Repository
2. The integration developer creates and exports a mediation module by completing the following steps.
   a. Create and configure a mediation module containing an Endpoint Lookup mediation primitive. Use Integration Designer to create a mediation module containing an Endpoint Lookup mediation primitive that refers to your WSRR registry. Creating mediation modules
   b. Define the endpoint selection policy that should be implemented. If you want to select a single service endpoint using the Endpoint Lookup mediation primitive, choose a `Match Policy` of `Return first matching endpoint and set routing target`. If you want to use service-version-aware routing, choose a `Match Policy` of `Return endpoint matching latest compatible service version`. Endpoint Lookup mediation primitive
   c. Optional: If you want to use service-version-aware routing, there are certain properties you must set on the Endpoint Lookup. Service-version-aware routing makes use of endpoint information that is in an SCA module (not in a WSDL document). Therefore, you must set the `Version` property to be the version of the SCA module. The asterisk (*) wildcard character is supported for the version number (for example, 1.1.*). You must also set the `Module` property. The module name must be the original name of the module, not the name that contains the version number or unique cell ID. Endpoint Lookup mediation primitive
   d. Ensure that the run time uses dynamic endpoint selection for your mediation module. From Integration Designer set the `Use dynamic endpoint`

property in the callout node. In order for the run time to implement dynamic routing on a request, you must set the `Use dynamic endpoint` property in the callout node.

e. Optional: Ensure that you can control dynamic endpoint selection from the runtime administrative console. Promote the `Use dynamic endpoint` property. Integration Designer lets you promote properties so they are visible from the runtime administrative console as module properties. You give alias names to promoted properties; and it is the alias names that are displayed on the runtime administrative console. Promoting a property in the mediation flow editor

f. Optional: Set a default endpoint. You can specify a default endpoint that the run time uses if it cannot find a dynamic endpoint. You specify a default endpoint by wiring a reference to an import.

g. Optional: Wire another mediation primitive to the Endpoint Lookup mediation primitive. If the Endpoint Lookup mediation primitive has a `Match Policy` of `Return all matching endpoints`, you must use another mediation primitive to select one particular endpoint. If the Endpoint Lookup mediation primitive has a `Match Policy` of `Return all matching endpoints and set alternate routing targets`, you do not need another mediation primitive to select a particular endpoint: by default the first match is used for the dynamic endpoint. Mediation primitives

h. Export the mediation module. If the mediation module is not versioned, you can export it as an EAR file. Otherwise, export it for command-line deployment and use the **serviceDeploy** command-line utility to build a deployable EAR file. Mediation primitives

3. The WSRR administrator loads the documents that contain the service endpoint information, into WSRR.

   a. Load the WSDL documents or SCA modules into WSRR. If an SCA module contains versioning information, WSRR can be queried to obtain service endpoints that relate to a version of the module. Managing services

4. The administrator creates a WSRR definition, and deploys and administers the mediation module by completing the following steps.

   a. If necessary, create and configure a registry definition. Use the runtime administrative console to configure a WSRR definition. Administering access to WSRR

   b. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. If you have promoted the `Use dynamic endpoint` property, you can change its value during deployment. Installing an SCA module with the console

   c. Optional: Ensure that dynamic endpoint selection is enabled. If you have promoted the `Use dynamic endpoint` property, check that it is set using the runtime administrative console. Displaying service module properties

   d. Start the application associated with the mediation module Use the runtime administrative console to start the application associated with the SCA module Starting and stopping service modules

Results

When a service request invokes your mediation module, the service request will be routed to the endpoint that matches your specification.

Example

Figure 96. Dynamic Routing using the Endpoint Lookup mediation primitive

Dynamic routing using a database

Choose which service is used at run time, using a database to lookup service endpoints.

About this task

If your goal is to have service requests routed to endpoints that are selected dynamically, you can achieve this by using a database combined with a Database Lookup mediation primitive. The run time supports dynamic endpoint selection, but you must set the `Use dynamic endpoint` property in your mediation module.

Figure 97. Actor (service requester) and system (ESB)

To dynamically route services complete the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module or a suitable database. You should refer to product documentation for these instructions.

Procedure

1. The database administrator creates and configures a database to hold your service endpoint information. Ensure that the database can be queried to return service endpoints.
2. The integration developer creates and exports a mediation module by completing the following steps.
   a. Create and configure a mediation module containing a Database Lookup mediation primitive. Use Integration Designer to create a mediation module containing a Database Lookup mediation primitive that refers to your database. Creating mediation modules
   b. Ensure that the service endpoint is found at run time. If you want to select a service endpoint using the Database Lookup mediation primitive alone, ensure that the `Message element` property is set to `/headers/SMOHeader/Target/address` because this is where the run time looks for the dynamic endpoint. Database Lookup mediation primitive
   c. Ensure that the run time uses dynamic endpoint selection for your mediation module. From Integration Designer set the `Use dynamic endpoint` property in the callout node. In order for the run time to implement dynamic routing on a request, you must set the `Use dynamic endpoint` property in the callout node.
   d. Optional: Ensure that you can control dynamic endpoint selection from the runtime administrative console. Promote the `Use dynamic endpoint` property. Integration Designer lets you promote properties so they are visible from the runtime administrative console as module properties. You give alias names to promoted properties; and it is the alias names that are displayed on the administrative console.
   e. Set a default endpoint. You can specify a default endpoint that the run time uses if it cannot find a dynamic endpoint. You specify a default endpoint by wiring the reference to an import that has the default service selected.
   f. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files
3. The runtime administrator deploys and administers the mediation module by completing the following steps.
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. If you have promoted the `Use dynamic endpoint` property, you can change its value during deployment. Installing an SCA module with the console
   b. Optional: Ensure that dynamic endpoint selection is enabled. If you have promoted the `Use dynamic endpoint` property, check that it is set using the administrative console. Displaying service module properties
   c. Start the application associated with the mediation module Use the runtime administrative console to start the application associated with the SCA module Starting and stopping service modules

Results

When a service request invokes your mediation module, the service request will be routed to the endpoint that matches your specification.

Example

Figure 98. Dynamic Routing using the Database Lookup mediation primitive

Dynamic routing using message filtering

Choose which service is used at run time, by filtering a message.

About this task

If your goal is to have service requests routed to one of a predefined set of endpoints, you can achieve this by filtering your messages using the Message Filter mediation primitive.

Figure 99. Actor (service requester) and system (ESB)

To dynamically route services complete the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module. You should refer to the Integration Designer for these instructions.

Procedure

1. The integration developer creates and exports a mediation module by completing the following steps.
   a. Create and configure a mediation module containing a Message Filter mediation primitive. Use Integration Designer to create a mediation module. Creating mediation modules
   b. Define the filtering that should be performed. Update the Message Filter properties to define the filter patterns against which input messages are tested. You can define a number of filter patterns. Message Filter mediation primitive
   c. Define the endpoints that the filters should use. You can route messages that meet the same filter pattern to different endpoints, or you can route messages that meet different filter patterns to the same endpoint. You determine whether messages can be sent to multiple endpoints or one endpoint, by setting the `Distribution mode` property.
   d. Optional: Ensure that you can adjust aspects of your routing, from the administrative console. Promote the properties that you want to control. Integration Designer lets you promote mediation primitive properties so they are visible from the runtime administrative console as module properties. You give alias names to promoted properties; and it is the alias names that are displayed on the runtime administrative console. com.ibm.wbpm.wid.integ.doc/topics/tmarkprom.html
   e. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files
2. The runtime administrator deploys and administers the mediation module by completing the following steps.
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. Installing an SCA module with the console
   b. Optional: Check that your mediation module has the correct properties. You can check and set the values of promoted properties using the administrative console. Displaying service module properties
   c. Start the application associated with the mediation module. Use the runtime administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When a service request invokes your mediation module, the service request will be routed based upon your filtering criteria.

Example

Figure 100. Dynamic Routing using the Message Filter mediation primitive

Dynamic routing of responses

You can choose which client application is sent a response at runtime, by using a database to lookup service endpoints.

About this task

If your goal is to have service responses redirected to endpoints that are selected dynamically, you can achieve this by using a database combined with a Database Lookup mediation primitive. The runtime supports dynamic response redirection, but you must set the `Use dynamic endpoint` property on the relevant `Input response` node.

Figure 101. Actor (service requester) and system (ESB)

To dynamically redirect responses complete the following steps.

Note: Other means of determining the correct redirected response endpoint address can be used (for example, using a registry lookup). It is also possible to

indicate that, in the case where the response flow has dealt with the response in some other manner, no response is to be sent by the export by using the target address: `http://www.w3.org/2005/08/addressing/none`.

Procedure

1. The database administrator creates and configures a database to hold your service endpoint address information (URLs). Ensure that the database can be queried to return service endpoint address URLs.
2. The integration developer uses Integration Designer to create and export a mediation module by completing the following steps:
   a. Create and configure a mediation module. In the mediation component, the response mediation flow should contain a Database Lookup mediation primitive. Use Integration Designer to create a mediation module containing a Database Lookup mediation primitive that refers to your database.
   b. Ensure that the redirected response endpoint is found at runtime. If you want to select a response endpoint using the Database Lookup mediation primitive alone, ensure that the `Message element` property is set to `/headers/SMOHeader/Target/address` because this is where the run time looks for the dynamic response endpoint address.
   c. Ensure that the runtime uses dynamic endpoint selection for your mediation module. From Integration Designer set the `Use dynamic endpoint` property in the `Input response` node. In order for the runtime to implement dynamic response redirection, you must set the `Use dynamic endpoint` property in the `Input response` node.
   d. Ensure that you can control dynamic endpoint selection from the runtime administrative console. Promote the `Use dynamic endpoint` property with Integration Designer. You can promote properties so they are visible from the runtime administrative console as module properties. You give alias names to promoted properties, and it is the alias names that are displayed on the administrative console.
   e. Export the mediation module. From Integration Designer export the mediation module as an EAR file.
3. The runtime administrator deploys and administers the mediation module by completing the following steps:
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. If you have promoted the `Use dynamic endpoint` property, you can change its value during deployment.
   b. Ensure that dynamic endpoint selection is enabled. If you have promoted the `Use dynamic endpoint` property, check that it is set using the administrative console.
   c. Start the application associated with the mediation module. Use the runtime administrative console to start the application associated with the SCA module.

Results

When a service request invokes your mediation module, the service response is routed to the endpoint that matches your specification.

Example

Figure 102. The following figure shows a Dynamic Response Redirection when using the Database Lookup mediation primitive

Mediation policy control of service requests

You can use mediation policies to control mediation flows between service requesters and service providers.

You can control mediation flows using mediation policies stored in IBM WebSphere Service Registry and Repository (WSRR). The implementation of service policy management in WSRR is based on the Web Services Policy Framework (WS-Policy).

In order to control service requests using mediation policies, you must have suitable Service Component Architecture (SCA) modules and mediation policy documents in your WSRR registry.

How to attach a mediation policy to a service request

When you develop an SCA module that needs to make use of a mediation policy, you must include a Policy Resolution mediation primitive in the mediation flow. At run time, the Policy Resolution mediation primitive obtains mediation policy information from the registry. Therefore, an SCA module must contain a mediation flow component in order to support mediation policy control of service requests.

In the registry, you can attach one or more mediation policies to an SCA module, or to a target service used by the SCA module. Attached mediation policies could be used (are in scope) for all service messages processed by that SCA module. The mediation policies can have policy attachments that define conditions. Mediation policy conditions allow different mediation policies to apply in different contexts. In addition, mediation policies can have classifications, which can be used to specify a governance state.

Controlling message flows using mediation policies

You can control message flows using mediation policies, and associated conditions.

Before you begin

Read the mediation policies scenario which provides an overview of mediation policies, and explains how the integration developer, the runtime administrator, and the WSRR administrator must work together.

About this task

If your goal is to use context information to control the behavior of service requests, you can achieve this by using mediation policies stored in WebSphere Service Registry and Repository (WSRR).

You can use mediation policies on either the request flow or the response flow, or both. Mediation policy information obtained on the request flow can be propagated to the response flow.

To help the explanation of mediation policies an example application is referred to. The example application is a ticket discount system in which the discount applied depends on the mediation policy used. The request flow extracts discount policies, based on the age of the traveller, and the service provider supplies the cost of a standard ticket for the journey. The response flow calculates the overall cost to the traveler, based on the discount supplied by the mediation policy.

Figure 103. Actor (service requester) and system (ESB)

To control the results of a mediation flow by using mediation policies, complete the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module or a WSRR registry. You should refer to the product documentation for these instructions.

Procedure

1. Optional: If necessary, the WSRR administrator creates and configures a WSRR registry. Installing Websphere Service Registry and Repository

2. The integration developer creates, configures, and exports a mediation module by completing the following steps.
   a. Create a mediation module. Creating mediation modules
   b. Complete the export and import configuration, generate an implementation, and connect a source operation to a target operation.
   c. On the request flow, add a Policy Resolution mediation primitive. When an SCA module contains a Policy Resolution mediation primitive, WebSphere Integration Developer generates default mediation policies when you export the module as an EAR file. Mediation primitives
   d. On the request flow, add any other mediation primitives you need, and wire the elements of the request flow together. Mediation primitives
   e. Optional: On the request flow, configure the Policy Resolution mediation primitive so that it can make use of a gate condition on a mediation policy. Create a **Policy condition name**, and provide the **XPath**: the location from which the run time gets the mediation policy condition. For example, if the **XPath** is set to `/body/priceChecker/request/Age` and the associated **Policy condition name** is Age, the run time sets the value of Age to whatever it finds at `/body/priceChecker/request/Age`. Policy Resolution mediation primitive
   f. On the request flow, configure the Policy Resolution mediation primitive so that the mediation policy information you get on the request flow, is available to the response flow. Select the **Propagate mediation policy to response flow** check box.
   g. On the response flow, add the mediation primitives whose dynamic properties you want to override with mediation policy information; wire the elements of the response flow together. Because a promoted property is also a dynamic property, its value can be overridden using mediation policies.
   h. On the response flow, create and promote the mediation primitive properties that you want to override. For example, on the Custom Mediation primitive you could create a new user property called *Discount*. Defining your own properties in a custom mediation primitive
   i. On the response flow, complete the mediation primitive configuration. For example, on the Custom Mediation primitive you could add Java code to apply the discount provided by the mediation policy, to the ticket price returned by the service provider. Creating Java code in a custom mediation primitive
   j. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files
3. The administrator creates a WSRR definition, and deploys and administers the mediation module by completing the following steps.
   a. Optional: If necessary, create and configure a registry definition. Use the runtime administrative console to configure a WSRR definition. Administering access to WSRR
   b. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. Installing an SCA module with the console
4. The WSRR administrator loads the SCA module (mediation module) into WSRR, and attaches mediation policies to the SCA module. (When an SCA module is loaded, any default mediation policies are automatically loaded.) Advanced tutorial: Creating mediation policies for modules
   a. Load the SCA module into WSRR. You select the EAR file containing the SCA module that you want to load.

b. Optional: Create your own mediation policies. For example, you might create two mediation policies, one for people over 59 years old, and one for people under 12 years old. Both mediation policies could specify an assertion for the dynamic property *Discount*, because they both want to override the value of *Discount*.
c. Attach the mediation policies that you want to use, to the SCA module. Mediation policies must be attached to SCA modules before they can be used by the run time, and before you can create gate conditions for them.
d. Optional: Create gate conditions for your own mediation policies. For example, one mediation policy could have a gate condition that specified `Age > 59`, and another mediation policy could have a gate condition that specified `Age < 12`.

5. The administrator enables the application.
   a. Start the application associated with the mediation module. Use the runtime administrative console to start the application associated with the SCA module. Starting and stopping service modules
   b. Optional: Access any registry changes. If the WSRR administrator updates the mediation policies and you want to have access to the changes immediately, you should clear the cache associated with the WSRR definition. Clearing the cache associated with a WSRR definition

Results

When a service request invokes your mediation module, the service request causes the registry to be queried. If suitable mediation policy information (dynamic property values) is returned, it is kept for the response flow. The service is then invoked and it sends a response. During the processing of the response flow, the mediation policy information is used to override the response values. Therefore, the response that the service requester receives depends on the context of the message.

Example

Figure 104. Controlling the behavior of service requests using mediation policies

Protocol conversion

Connect service endpoints that use different protocols.

About this task

If your goal is to connect endpoints that use different protocols you can achieve this by creating mediation modules that convert between different protocols. For example, if you want to connect Web services using SOAP/HTTP to Java Message Service (JMS) applications.

Figure 105. Actor (service requester) and system (ESB)

To connect endpoints that use different protocols complete the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module. For complete instructions on how to construct a mediation module refer to the Integration Designer documentation.

Procedure

1. The integration developer creates and exports a mediation module by completing the following steps.
   a. Create and configure a mediation module. Use Integration Designer to create a mediation module. Creating mediation modules
   b. Create an export with a binding appropriate to the service requester. If the service requester is a Web service, the binding can be a *Web service binding*, (using SOAP/HTTP or SOAP/JMS), or an *HTTP binding*.
      - For HTTP bindings, see: Creating exports
      - For Web Service bindings, see: Generating Web service bindings for exports
   c. Create an import with a binding appropriate to the service provider. If the service provider is a JMS application, the binding should be a JMS binding. The run time supports different types of JMS bindings in order to support different JMS providers. Generating a JMS import binding
   d. Wire the mediation module. If two services have the same interface, and no mediations are required, you can wire the export directly to the import. If two services have different interfaces, or you need to mediate the message, a mediation flow component is needed to do the transformations.
   e. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files
2. The runtime administrator deploys and administers the mediation module by completing the following steps.
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. Installing an SCA module with the console
   b. Start the application associated with the mediation module. Use the runtime administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When a service request invokes your mediation module then messages are converted from one protocol to another by the run time.

Example

Figure 106. Message protocol conversion using a mediation module

Validating messages

Validate messages against their schema.

About this task

If your goal is to validate messages against their schema, you can achieve this by creating mediation modules that contain the `Validate input` property.

Figure 107. Actor (service requester) and system (ESB)

To validate messages against their schema complete the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module. For complete instructions on how to construct a mediation module refer to the Integration Designer documentation.

Procedure

1. The integration developer creates and exports a mediation module by completing the following steps.
   a. Create and configure a mediation module containing a mediation primitive with the `Validate input` property. Use Integration Designer to create a mediation module. Creating mediation modules
   b. Ensure that you can control message validation from the runtime administrative console. Promote the `Validate input` property. Integration Designer lets you promote mediation primitive properties so they are visible from the runtime administrative console as module properties. You give alias names to promoted properties; and it is the alias names that are displayed on the runtime administrative console. Promoting a property in the mediation flow editor
   c. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files
2. The runtime administrator deploys and administers the mediation module by completing the following steps.
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. If you have promoted the `Validate input` property, you can change its value during deployment. Installing an SCA module with the console
   b. Check that message validation is set. You can check and set the values of promoted properties using the runtime administrative console. You can control message validation by turning the `Validate input` property on and off. Displaying service module properties
   c. Start the application associated with the mediation module. Use the administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When a service request invokes your mediation module, messages are validated against their schemas at the points where validation is requested.

Example

Figure 108. Message validation using the Database Lookup and Message Element Setter mediation primitives

Connect to CICS or IMS services using MQ

You can connect to CICS or IMS services using WebSphere MQ, by receiving messages from, or sending messages to, the corresponding MQ Bridge.

You can connect your ESB to CICS or IMS services using MQ. For example, you can receive and process messages from CICS using MQ, or you can create and send a message to IMS using MQ.

When your ESB connects to CICS or IMS, you might need access to information carried in the MQ header structures defined by the MQ CICS bridge, or the MQ IMS bridge. This information is carried in the MQCIH for CICS, or the MQIIH for IMS. You can access this information by using the Custom Mediation primitive, or the MQ Header Setter mediation primitive. You can also create or modify header information.

Calling a CICS service

You can connect to a CICS service and send a message through the MQ-CICS Bridge, with dynamic header information.

About this task

If your goal is to connect your ESB to a CICS service using WebSphere MQ, you might need to create information carried in the MQ header structures defined by the MQ-CICS Bridge. This information is carried in the MQCIH. You can create this information dynamically by using the MQ Header Setter mediation primitive, or statically by setting MQImport binding properties.

To create MQCIH information using the MQ Header Setter mediation primitive, complete the following steps.

Note: These steps provide an overview, they do not include all instructions for creating a complete mediation module. For complete instructions on how to construct a mediation module refer to the Integration Designer documentation.

Procedure

1. The integration developer creates and exports a mediation module by completing the following steps.
   a. Create and configure a mediation module containing an MQ Header Setter mediation primitive. Use Integration Designer to create a mediation module. Creating mediation modules
   b. Add an MQ Header Element. Follow any instructions to add a dependency, from the module to the MQ schemas. Then set the MQ Header Setter properties in the following way:
      - *Header Action*: `Create`.
      - *Header Type*: `MQCIH`.
      - *Set Values*: set the values that you need for the `MQCIH` fields.

      MQ Header Setter mediation primitive
   c. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files
2. The runtime administrator deploys and administers the mediation module by completing the following steps.
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. If you have promoted the `Validate input` property, you can change its value during deployment. Installing an SCA module with the console
   b. Start the application associated with the mediation module. Use the administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When a service request invokes your mediation module, a new MQCIH header element is added to the message.

Accessing MQCIH headers

MQCIH headers, in a message received from CICS through the MQ-CICS Bridge, can be read using the MQ Header Setter mediation primitive.

About this task

To access MQCIH information in a message received from CICS through the MQ-CICS Bridge, use the MQ Header Setter mediation primitive by completing the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module. For complete instructions on how to construct a mediation module, refer to the Integration Designer documentation.

Procedure

1. The integration developer creates and exports a mediation module by completing the following steps.
   a. Create and configure a mediation module containing an MQ Header Setter mediation primitive. Use Integration Designer to create a mediation module. Creating mediation modules

b. Add an MQ Header Element. Follow any instructions to add a dependency, from the module to the MQ schemas. Then set the MQ Header Setter properties in the following way:
   - *Header Action*: `Find and Copy`.
   - *Header Type*: `MQCIH`.
   - *Target Destination*: XPath expression defining where the MQCIH header should be copied to. After the header has been copied to the target destination, later mediation primitives can inspect or process it.

   MQ Header Setter mediation primitive

c. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files

2. The runtime administrator deploys and administers the mediation module by completing the following steps.
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. If you have promoted the `Validate input` property, you can change its value during deployment. Installing an SCA module with the console
   b. Check that message validation is set. You can check and set the values of promoted properties using the runtime administrative console. You can control message validation by turning the `Validate input` property on and off. Displaying service module properties
   c. Start the application associated with the mediation module. Use the administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When a service request invokes your mediation module, the MQCIH header is copied to the target element specified by the *Target Destination* XPath expression.

Calling an IMS service

You can connect to an IMS service and send a message through the MQ-IMS Bridge.

About this task

If your goal is to connect your ESB to an IMS service using WebSphere MQ, you might need to create information carried in the MQ header structures defined by the MQ IMS bridge. This information is carried in the MQIIH. You can create this information dynamically by using the MQ Header Setter mediation primitive, or statically by setting MQImport binding properties.

To create MQIIH information using the MQ Header Setter mediation primitive, complete the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module. For complete instructions on how to construct a mediation module refer to the Integration Designer documentation.

Procedure

1. The integration developer creates and exports a mediation module by completing the following steps.

a. Create and configure a mediation module containing an MQ Header Setter mediation primitive. Use Integration Designer to create a mediation module. Creating mediation modules
b. Add an MQ Header Element. Follow any instructions to add a dependency, from the module to the MQ schemas. Then set the MQ Header Setter properties in the following way:
   - *Header Action*: `Create`.
   - *Header Type*: `MQIIH`.
   - *Set Values*: set the values that you need for the `MQIIH` fields.

   MQ Header Setter mediation primitive
c. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files

2. The runtime administrator deploys and administers the mediation module by completing the following steps.
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. If you have promoted the `Validate input` property, you can change its value during deployment. Installing an SCA module with the console
   b. Start the application associated with the mediation module. Use the administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When a service request invokes your mediation module, a new MQIIH header element is added to the message.

Accessing MQIIH headers

MQIIH headers, in a message received from IMS through the MQ-IMS Bridge, can be read using the MQ Header Setter mediation primitive.

About this task

To access MQIIH information in a message received from IMS through the MQ-IMS Bridge, use the MQ Header Setter mediation primitive by completing the following steps.

Note: these steps provide an overview, they do not include all instructions for creating a complete mediation module. For complete instructions on how to construct a mediation module, refer to the Integration Designer documentation.

Procedure

1. The integration developer creates and exports a mediation module by completing the following steps.
   a. Create and configure a mediation module containing an MQ Header Setter mediation primitive. Use Integration Designer to create a mediation module. Creating mediation modules
   b. Add an MQ Header Element. Follow any instructions to add a dependency, from the module to the MQ schemas. Then set the MQ Header Setter properties in the following way:
      - *Header Action*: `Find and Copy`.

- *Header Type*: `MQIIH`.
- *Target Destination*: XPath expression defining where the MQIIH header should be copied to. After the header has been copied to the target destination, later mediation primitives can inspect or process it.

MQ Header Setter mediation primitive

c. Export the mediation module. From Integration Designer export the mediation module as an EAR file. Exporting modules as EAR files

2. The runtime administrator deploys and administers the mediation module by completing the following steps.
   a. Deploy the mediation module. Use the runtime administrative console to install the EAR file as a new application. If you have promoted the `Validate input` property, you can change its value during deployment. Installing an SCA module with the console
   b. Check that message validation is set. You can check and set the values of promoted properties using the runtime administrative console. You can control message validation by turning the `Validate input` property on and off. Displaying service module properties
   c. Start the application associated with the mediation module. Use the administrative console to start the application associated with the SCA module (mediation modules are types of SCA modules). Starting and stopping service modules

Results

When a service request invokes your mediation module, the MQIIH header is copied to the target element specified by the *Target Destination* XPath expression.

Chapter 4. Tutorials for WebSphere ESB

Tutorials for WebSphere ESB provide information designed around specified learning objectives. A tutorial is often broken into smaller chunks of information, called modules and lessons. A tutorial teaches fundamental distinctions, skills, and approaches to using a product or technology more effectively.

Besides tutorials provided in this documentation, additional tutorials are available in the IBM Education Assistant.

Related information:

PDF documentation
WebSphere Enterprise Service Bus documentation (in PDF format)

Information roadmaps
Business Process Management information roadmaps on IBM developerWorks organize information about WebSphere Business Process Manager, WebSphere ESB, and the other products in the portfolio.

IBM Education Assistant
Multimedia educational modules about WebSphere ESB, provided by IBM Education Assistant.

Overview
Overview tab, on product library Web page. Use this page to access announcements, data sheets, and other general library documents related to WebSphere ESB.

Advanced tutorial: Creating mediation policies for modules

You can use mediation policies to control message flows between service requesters and service providers. Mediation policies can be associated either with modules or with target services: this tutorial deals with mediation policies associated with modules. This tutorial uses WebSphere Service Registry and Repository (WSRR), not Business Space widgets.

This tutorial shows you how to configure WSRR, so that you can use mediation policies to control your message flows. The tutorial is broken into smaller chunks of information called education modules, and each education module has one or more lessons.

This tutorial shows you how to load an SCA module and associated mediation policies into WebSphere Service Registry and Repository (WSRR); also, how to attach a mediation policy to your SCA module.

This tutorial is classified as advanced because it uses the WSRR Web user interface, rather than Business Space widgets. For more information on using widgets to create mediation policies, see Tutorial: Using widgets to create mediation policies.

Notes:

- You can use Business Space widgets to create mediation policies and conditions. However, you cannot use Business Space widgets to load SCA modules into WSRR, or to govern your policies.

© Copyright IBM Corp. 2007, 2011

- You can use this tutorial if you want to implement the mediation policy pattern *Mutually exclusive gate conditions*. If you want to use the simplest mediation policy pattern, *Default mediation policies only*, you need only do education modules one, two, and four. For more information about mediation policy patterns, see: Mediation policy patterns.
- You need only read the education module on governance if you want to use WSRR governance.

Introduction

Mediation policies can be associated with SCA modules, and let you control the dynamic properties of mediations.

When you use IBM Integration Designer to create SCA modules that contain mediation flows, any module property that you promote (make visible to the run time) is also a dynamic property. Dynamic properties can be overridden, at run time, using mediation policies in a registry.

When you export your SCA module, IBM Integration Designer generates a default mediation policy for each property group in your SCA module. The default mediation policies represent the values given to all dynamic properties, at development time. Although IBM Integration Designer generates default mediation policies, it does not attach them to the SCA module.

Generally, use default mediation policies without any associated conditions, and create your own mediation policies if you want to have conditional mediation policies.

Learning objectives

- Understand how SCA modules and mediation policies relate to each other.
- Learn how to load SCA modules, and system-defined mediation policies, into WSRR.
- Learn how to create a mediation policy.
- Learn how to attach a mediation policy to an SCA module.
- Learn how to create gate conditions for a mediation policy.

This tutorial takes approximately 1 hour to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Skill level

Advanced.

Audience

Integration Developers.

Administrators.

System requirements

- IBM Integration Designer, Version 7.5
- WebSphere Enterprise Service Bus, Version 7 or IBM Business Process Manager, Version 7.5
- WebSphere Service Registry and Repository, Version 7.5

Prerequisites

1. Read the mediation policies scenario which explains the overview, and also explains how the developer, the runtime administrator, and the WSRR administrator must work together.
2. Log on to the WSRR Web user interface. To start the Web user interface, enter the following URL into the address bar of the Web browser: `http://hostname:port/ServiceRegistry` where `hostname` is the IP address or host name of the WebSphere Application Server that is hosting the WSRR application, and `port` is the server port number.
3. If necessary, switch to the governance enablement profile. By default, WSRR Version 7 uses a governance enablement profile; therefore, this tutorial assumes that you are using the governance enablement profile. To check which profile is active, click **Perspective** > **Configuration**, then click **Manage Profiles** > **Configuration Profiles**.
4. If necessary, switch to the WSRR Administrator perspective: click **Perspective** > **Administrator**. This tutorial assumes that you are using the Administrator prespective.

Loading an SCA module and associated mediation policies

Mediation policy control is achieved at the level of the SCA module. This education module shows you how to load an SCA module into WSRR, so you can attach a mediation policy to it.

Learn how to load an SCA module into WSRR.

5 minutes.

Loading an SCA module

Load an SCA module, in an EAR file, from the local file system. If the EAR file contains mediation policies, the mediation policies are also loaded.

If you have a versioned SCA module that you need to deploy to clusters, run **`CreateVersionedSCAModule`** to create an EAR file that contains a unique instance of a versioned SCA module. Create one instance of the module for every cluster that you want to deploy to, and load your versioned SCA modules into WSRR from the new EAR files.

When you load the EAR file containing your SCA module, WSRR creates an SCA module document. WSRR also loads any system-generated mediation policies.

1. In the WSRR Web user interface, click **View** > **Service Documents** > **SCA Integration Modules**. The Load Documents page is displayed.
2. Click **Load Documents**. The Load Documents page is displayed.
3. Click **Browse**, to find the SCA integration module that you want to load.
4. Select the EAR file containing the SCA module that you want to load. The path to the EAR file is displayed.
5. Enter the document description.
6. Optional: Enter the version. The version referred to on this panel is the WSRR document version, not the SCA module version (every WSRR document can have a version identifier). If you load a versioned SCA module, the runtime registry query needs to find the correctly versioned SCA module (the WSRR document version is not relevant, in this context). If you want, you can keep the SCA module version the same as the WSRR document version. To keep the version identifiers the same you must manually enter the SCA module number.

For example, if your EAR file is called CreditHistory_v1_0_0App.ear and your SCA module is called CreditHistory_v1_0_0 enter the version number: v1_0_0.

Note: You cannot change the WSRR version identifier after you have loaded your SCA module.

7. Click **OK**.

WSRR creates an SCA module document.

WSRR also loads any default mediation policies that were generated by IBM Integration Designer. If you want to provide a dynamic override for every dynamic property in your module, you can attach all the default (system-generated) mediation policies to your SCA module.

Example: Loading an SCA integration module

Viewing a mediation policy

When you load an SCA integration module into WSRR, any system-generated (default) mediation policies are loaded along with the SCA module. This education module shows you how to view mediation policies in general, and how to identify the default mediation policies.

Learn how to view a mediation policy, and how to identify mediation policies that have been generated by IBM Integration Designer.

5 minutes.

Load the SCA module containing the default mediation policies.

Viewing a default mediation policy

View the default mediation policies, generated for your SCA module by IBM Integration Designer.

An SCA integration module that contains a Policy Resolution mediation primitive, generally contains one or more default mediation policies generated by IBM Integration Designer. There is one default mediation policy for each property group. However, these system-generated mediation policies are not automatically attached to the associated SCA module. You must decide which dynamic properties you want to override and, therefore, which mediation policies to attach to your SCA module.

Note: Any dynamic properties in the SCA module appear as mediation policy assertions, in the mediation policy document created by WSRR.

A mediation policy has a namespace, which is made up of the group name of the property appended to: `http://www.ibm.com/wbi/mediation/200812/`. The integration developer can specify the property group and the administrative console displays module properties in their property groups. The namespace is displayed by WSRR as an additional property of the mediation policy document, called xmlns_sibx.

1. In the WSRR Web user interface, click **View** > **Service Documents** > **Policy Documents**. The Policy Documents page is displayed.
2. Click a mediation policy document. Generally, mediation policy documents that have been system-generated have a name of the format: *application-group*. For example, if the EAR file were called SimpleApp.ear and the groups were called groupA and groupB, the mediation policy documents might be called SimpleApp-groupA.xml and SimpleApp-groupB.xml.
3. Click the **Policy** tab.

You can now see the contents of the mediation policy document. For default mediation policies, each assertion relates to a dynamic property. When a mediation policy is used at run time, its assertion values override any dynamic properties in the mediation flow.

Example: Viewing a mediation policy for an SCA module

The following figure shows a mediation policy document with three properties for groupA. The property names are: propA1, propA2, and propA3.

Figure 109. Mediation policy document for groupA of SimpleApp

The following figure shows the same three properties, for groupA, seen from the administrative console. You can adjust the values from the administrative console. However, if the appropriate mediation policy is attached, and used, the mediation policy values take precedence.

Figure 110. Properties for SCA module SimpleApp

Creating your own mediation policy, for an SCA module

If you want to do more than override all the dynamic properties of an SCA module, you can create your own mediation policies, in WSRR. This education module shows you how to create a mediation policy.

Learn how to create a mediation policy.

15 minutes.

Creating a mediation policy

Create your own mediation policy.

If you want to override specific dynamic properties of an SCA module you can create your own mediation policy. (Generally, you would create your own mediation policy if you wanted to apply mediation policies conditionally; according to gate conditions.)

1. In the WSRR Web user interface, click **View** > **Service Documents** > **Policy Documents**. The Policy Documents page is displayed.
2. Click **New**.
3. Select **WS Policy Framework 1.5**.
4. Click **Next**.
5. Enter the name of the mediation policy.
6. Optional: Enter the description.
7. Click **Policy**.
8. Click **Add Property**. The Optional Properties field must say Policy Identifier.
9. Click **Add**.
10. Enter a unique identifier. The identifier must be unique for this instance of WSRR. Add the identifier after the *urn:*, for example *urn:UniqueIdentifier*.
11. Click **Select Policy Domain**. The policy domains are listed. All mediation policy domains have the form: Mediation Domain [xxx], where xxx is the domain name. For example, Mediation Domain [groupA].
12. Select the policy domain for your mediation policies.
13. Click **Apply**.
14. Click **Select Policy Type**. The policy type relates to the property group you defined in IBM Integration Designer.
15. Click **Select**.
16. Click **Add Assertion**.
17. Select a mediation policy assertion, for a property you want to override. Each assertion relates to a dynamic property in a particular property group. Therefore, you can select an assertion for any dynamic property in a particular property group.
18. Click **Add**.
19. Enter a value for the assertion.
20. Optional: Add any other assertions and values that you want to have in your mediation policy.

    Note: You do not have to give values to all assertions, you give values to the properties you want to override.
21. Click **Publish**.

WSRR creates a mediation policy that contains the property overrides you specified. Before the overrides can be used, you must attach the mediation policy to your SCA module.

Example: User-generated mediation policy for an SCA module

The following figure shows an assertion being added to a new mediation policy document. The mediation policy contains a groupA property (propA1) whose value is set to `All`.

Figure 111. Mediation policy for groupA of SimpleApp

Attaching a mediation policy to an SCA module

To associate particular mediation policies with a particular SCA module, you must attach the mediation policies to the SCA module. This education module shows you how to attach a mediation policy to an SCA module.

Learn how to attach a mediation policy to an SCA module, in WSRR.

10 minutes.

Attaching a mediation policy

Attach a mediation policy to an SCA module.

When you load an SCA integration module into WSRR, any default mediation policies are loaded, along with the SCA module. However, the default mediation policies are not attached to the SCA module. It is up to you to decide which mediation policies to use.

1. In the WSRR Web user interface, click **View** > **Service Documents** > **SCA Integration Modules** > **SCA Module Documents**. The SCA Module Documents page is displayed.
2. Select your SCA module document. The details of the SCA module document are displayed.
3. Under **Relationships** > **Modules**, select your module. The module details are now displayed.
4. Click the **Policy** tab. You can now see the mediation policy details.
5. Click **Edit Policy Attachments**.
6. Click **Attach Policy**. You can now see the Add Policy Attachment information.
7. Click **Search**.
8. Select the mediation policy you want to attach to your SCA module.
9. Click **Attach Selected Policy**.
10. Click **Finish**.

WSRR creates a policy attachment document, that attaches the selected mediation policy to your SCA module.

Example: Mediation policy attachments

The following figure shows one mediation policy attachment for the SCA module SimpleApp.

Figure 112. Mediation policy attachment for SimpleApp

Creating a conditional mediation policy

You can create gate conditions for your own mediation policies, in WSRR. This education module shows you how to create a gate condition on a mediation policy attachment. A mediation policy attachment attaches a mediation policy to an SCA module.

Learn how to create a condition for a mediation policy attachment, in WSRR.

10 minutes.

Creating a condition for a mediation policy

Create a gate condition for your mediation policy. The gate condition is created on the mediation policy attachment, in WSRR.

1. In IBM Integration Designer, create conditions on the Policy Resolution mediation primitive.
2. In WSRR, create your own mediation policy and attach it to an SCA module.

After attaching a mediation policy to your SCA module, you can create a gate condition on the mediation policy attachment that WSRR has generated.

You create gate conditions by adding user properties to the mediation policy attachment. The run time interprets some of the user properties as necessary conditions (gate conditions): the conditions must be met before the mediation policy can be used. Only user properties that begin with the string `medGate_` are used as gate conditions for mediation policies.

Generally, add gate conditions to mediation policies that you create yourself, do not add gate conditions to system-generated mediation policies.

1. In the WSRR Web user interface, click **View** > **Service Documents** > **Policy Documents** > **Documents with Policy Attachments**. The Documents with Policy Attachments page is displayed. This page lists all the policy attachment documents.
2. Click the mediation policy attachment document that you want to update. The details of the policy attachment document are displayed.
3. Under **Relationships** > **Attachments**, select your policy attachment. The details of the policy attachment object are displayed.
4. Click **Edit Properties**. You can now edit some of the properties of the policy attachment.
5. Click **Add Property**. You can now add a custom property name.
6. Enter the gate condition name. Gate condition names must start with `medGate_` followed by a user-defined string, for example, `medGate_Condition1`.
7. Click **Add**.
8. Enter the gate condition. The syntax of the gate condition is: *policy_condition_name* operation *gate_value*. Specify the *policy_condition_name* as a **`Policy condition name`** in the Policy Resolution mediation primitive. The operation must be one of the following values:
   - =
   - !=
   - >
   - <
   - <=
   - >=

   The *gate_value* is the value being compared. If you entered a gate condition of `age > 59`, you would have to specify `age` as a **`Policy condition name`** in the Policy Resolution mediation primitive.
9. Click **OK**.

WSRR creates the condition for your mediation policy.

In the mediation policy processing model, mediation policies with gate conditions have the highest precedence. Therefore, property values defined by these mediation policies have the highest precedence. Mediation policies without gate conditions have a lower precedence. Therefore, property values defined by these mediation policies have a lower precedence. Administrative console values are used if there is no suitable property in a mediation policy.

Example: Gate condition on a mediation policy attachment

The following figure shows the additional properties of a mediation policy attachment, after a gate condition has been added (but before the change has been applied). The gate condition is called *medGate_Condition1* and has a value of `age > 59`.

Figure 113. Gate condition on a mediation policy attachment

Governing a mediation policy

Using WSRR classification systems, including life cycle classifications, you can filter the visibility of mediation policies. This education module shows you how to govern mediation policies by using WSRR life cycle classifications.

Learn how to govern mediation policies, in WSRR.

15 minutes.

Discovering classification URIs

In order for a mediation policy to be visible at run time, any classifications you define at integration time must match classifications on the mediation policy at run time. You must use the classification URIs, shown in WSRR, when developing your SCA module.

Determine what governance states you need for your mediation policies, and whether you already have suitable life cycle states in WSRR. There is a default governance life cycle, which is a set of life cycle classifications. However, you can define your own life cycle and include the categories required by your organization.

WSRR supports many classification systems, including life cycle classifications that you can use to govern your mediation policies.

If you make your policy document governed, in WSRR, you can take advantage of the governance capabilities of WSRR. For example, when you make a policy document governed, the governance state is percolated to any associated policies. In addition to using classifications for WSRR governance, you might want to use other types of WSRR classifications, in which case you can add classifications directly to the relevant WSRR policies, rather than to the policy documents.

In WSRR, you must find the URI of any classification that you want to use. Then, in IBM Integration Designer, you must create a Policy Resolution **`Classification`** property using that URI.

In WSRR, you can find the URI of a classification in a number of ways. The following steps show you one way.

1. In the WSRR Web user interface, click **View** > **Service Documents** > **Policy Documents**. The Policy Documents page is displayed.
2. Click a mediation policy document. The mediation policy document is displayed.
3. Click **Edit Classifications**. The Classifications page is displayed.
4. If the policy document does not have the classification whose URI you want, perform the following steps to find a classification for the default life cycle:
   a. Expand the Governance Profile Lifecycle Classifications.
   b. Expand the State.
   c. Expand the Default Governance Life Cycle.
   d. Select the check box of the Default Governance Life Cycle state that you want to use.
   e. Click **Add >>>**. If your policy document already has the classification whose URI you want, this step is unnecessary.
5. Select (highlight) the classification whose URI you want. The URI, of the classification, appears in a table.
6. Copy the URI.
7. Required: Click **Cancel**. Do not add governance classifications using the **Edit Classifications** method.

You can now reuse the URI in the **`Classification`** property of a Policy Resolution mediation primitive.

Example: URI for a life cycle state

The following figure shows the URI for a life cycle state. You must use this URI when you develop your SCA module.

Figure 114. Classification URI

Making a mediation policy governed in WSRR

If you want to use WSRR governance, you have to make your policy document governed and then move it from state to state.

Determine what governance states you need for your mediation policies, and whether you already have suitable life cycle states in WSRR. There is a default governance life cycle (set of classifications), but you can define your own life cycle and include the categories required by your organization.

Any mediation policy, whether default or user-generated, can be governed in WSRR.

1. In the WSRR Web user interface, click **View** > **Service Documents** > **Policy Documents**. The Policy Documents page is displayed.
2. Click a mediation policy document. Generally, default (system-generated) mediation policy documents have a name of the format: *application-group*. For example, if the EAR file were called SimpleApp.ear and the groups were called

groupA and groupB, the mediation policy documents might be called SimpleApp-groupA.xml and SimpleApp-groupB.xml.

3. Click the **Governance** tab.
4. Click **Govern**. Having entities in a life cycle allows governance controls to be enforced (using the relevant classification).

   Note: A governed entity moves from one life cycle state to another, it does not have more than one life cycle state at a time.
5. Select a life cycle type from the Available state transitions. If you want to use the default life cycle select **Initiate Default Lifecycle**.
6. Optional: If you want to change the governance state, select an Available state transitions state and click **Transition**. If a policy document is governed and you move it to another life cycle state, any associated policies (relationship targets) have their life cycle classification state changed, as well. Associated policies have their life cycle classification state changed because they are in the same governed collection.

Your mediation policy is now governed, in WSRR.

Example: Governed mediation policy

The following figure shows a policy document after it has been made governed. The life cycle used was the default life cycle.

Figure 115. Governed mediation policy

Tutorial summary

The mediation policy tutorial shows you how to create WSRR documents for SCA modules and mediation policies. In addition, it shows you how to create gate conditions for your mediation policies.

If you have not yet configured your runtime environment, do so now. For example, ensure that you have a suitable WSRR definition for your SCA module.

After you have configured your runtime environment and completed this tutorial, you can run the application associated with your SCA module. Depending on the message being processed, and on any gate conditions and classifications, the mediation flow can be altered by mediation policies.

Tutorial: Building clustered topologies in WebSphere ESB

A deployment environment describes the topology that you can configure using WebSphere ESB. This tutorial guides you through the steps to create a deployment environment based on a pattern (the Remote Messaging and Remote Support pattern), which is the recommended topology for a production environment. The clustered topology has all of the functional components such as user applications, messaging infrastructure, Common Event Infrastructure (CEI), and support applications.

Note: The four-cluster topology was introduced in WebSphere ESB V7.0.0.3.

Before you begin

Before you begin make sure you have installed the following software:

- A network deployment topology configured in WebSphere ESB Version 7 or later.
- A supported version of DB2® (Version 8.2 feature pack 6 or Version 9.1).

About this task

The tasks described in this tutorial make up the procedure for creating a Network Deployment environment based on the Remote messaging and remote support pattern.

Learning objectives

At the end of this tutorial you will be able to create a clustered topology by completing the following steps:

1. Creating advanced deployment manager profile using the Profile Management Tool
2. Starting the deployment manager
3. Creating custom profiles (managed nodes) for the Deployment manager using PMT
4. Federating the nodes into the deployment manager cell
5. Creating the deployment environment using the Deployment Environment configuration wizard.

Creating advanced deployment manager profile using the Profile Management Tool

Learn how to use the Advanced option of the Profile Management Tool to create and configure WebSphere ESB deployment manager profiles. Selecting the Advanced option creates profiles with customized configuration settings.

Before you begin

The software is installed on your system.

Restrictions:

- You cannot use the Profile Management Tool to create or augment profiles on WebSphere ESB installations on 64-bit architectures except on the Linux on zSeries® platform. To create profiles on other 64-bit architectures, you can use the **manageprofiles** command-line utility. For information about using the **manageprofiles** command-line utility, see Using the manageprofiles command. You can also use the Profile Management Tool on these architectures if you use an WebSphere ESB 32-bit installation.
- Vista Windows 7 **Restriction for nonroot users with multiple instances:** If you install multiple instances of WebSphere ESB as the root user and give a nonroot user access to only a subset of those instances, the Profile Management Tool does not function correctly for the nonroot user. In addition, a `com.ibm.wsspi.profile.WSProfileException` or `Access is denied` message occurs in the *install_root*`\bin\ProfileManagement\pmt.bat` file. By default, nonroot users do not have access to the `Program Files` directory, which is the default installation location for the product. To resolve this issue, nonroot users can install the product or be given permission to access the other product instances.

Linux UNIX Windows The language of the Profile Management Tool is determined by the default language on the system. If the default language is not one of the supported languages, then English is used. You can override the default language by starting the Profile Management Tool from the command line and using the `java user.language` setting to replace the default language. Run the following command:

- Linux UNIX *install_root*`/java/bin/java -Duser.language=`*locale* *install_root*`/bin/ProfileManagement/startup.jar`
- Windows *install_root*`\java\bin\java -Duser.language=`*locale* *install_root*`\bin\ProfileManagement\startup.jar`

For example, to start the Profile Management Tool in the German language on a Linux system, type the following command:

```
install_root/java/bin/java -Duser.language=de install_root/  \
bin/ProfileManagement/startup.jar
```

About this task

This task describes how to create an advanced deployment environment profile using the profile management tool.

Procedure

1. Start the WebSphere ESB Profile Management Tool.

   Starting the tool on Windows platforms:

   Windows You can use the following methods to start the tool on Windows platforms:

   - Use the Windows Start menu. For example, click **Start > Programs** *or* **All Programs > IBM WebSphere > Enterprise Service Bus 7.0 > Profile Management Tool**.
   - Run the command *install_root*`\bin\ProfileManagement\pmt.bat`

Starting the tool on Linux and UNIX platforms

Linux You can start the tool on Linux and UNIX platforms by running the command *install_root*/bin/ProfileManagement/pmt.sh

On Linux platforms only, you can also use operating system menus to start the Profile Management Tool. For example, click *Linux_operating_system_menus_to_access_programs* > IBM WebSphere > *your_product* > Profile Management Tool

The Welcome page is displayed.

2. In the Welcome page, click the **Launch Profile Management Tool** button or the **Profile Management Tool** tab.

   The **Profiles** tab is displayed.

3. In the **Profiles** tab, click **Create**.

   The **Profiles** tab can contain a list of profiles that have been created on your machine. For this procedure, it is assumed you are creating a new profile, not augmenting an existing one.

   The Environment Selection page opens in a separate window.

4. In the Environment Selection page, expand WebSphere Enterprise Service Bus and select the type of profile you want to create. Then click **Next**

   The Profile Creation Options page is displayed.

5. In the Profile Creation Options page, choose to perform a an **Advanced** profile creation, and click **Next** to display the Optional Application Deployment page.

   The **Advanced** option lets you specify your own configuration values for a profile.

   By selecting the **Advanced** option, you can do the following:
   - Assign customized values to ports, to the location of the profile, and to the names of the profile, node, host, and cell (when applicable).
   - Configure the Common database.
   - Deploy the administrative console.
   - Enable administrative security.
   - Create a system service to run the server, if your operating system and the privileges of your user account permit the creation of services.

6. In the Optional Application Deployment page, select whether to deploy the administrative console to the profile environment you are creating, then click **Next** to display the Profile Name and Location page.

   The administrative console is a Web-based tool that manages the server. Leave the **Deploy the administrative console (recommended)** check box selected.

7. In the Profile Name and Location page, perform the following steps.

   a. Specify a unique name and directory path for the profile, or accept the defaults.

      Each profile that you create must have a name. When you have more than one profile, you can tell them apart at their highest level by this name. If you choose not to use the default name, see Naming considerations for profiles, nodes, servers, hosts, and cells for information about issues you must consider when naming the profile, such as restrictions on the length of the directory name.

      The directory you specify will contain the files that define the runtime environment, such as commands, configuration files, and log files. By default, this directory location is:

      - Linux UNIX *install_root*/profiles/*profile_name*
      - Windows *install_root*\profiles*profile_name*

where *profile_name* is the name you specified. An error message is displayed if:

- The *profile_name* you specify is not unique.
- The directory you specify is not empty.
- Your user ID does not have sufficient permissions for the directory.
- There is insufficient space to create the profile.

b. You can make the profile that you are creating the default profile (so that commands work automatically with it) by selecting the **Make this profile the default** check box. This check box appears only if you have an existing profile on your system.

   The first profile that you create on a workstation is the default profile.

   The default profile is the default target for commands that are issued from the `bin` directory in the product installation root. When only one profile exists on a workstation, every command operates on that profile. If more than one profile exists, certain commands require that you specify the profile to which the command applies. See Profile commands in a multi-profile environment for more information.

c. Click **Next**. (If you click **Back** and change the name of the profile, you might have to manually change the name on this page when it is displayed again.)

   The Node, Host, and Cell Names page is displayed.

8. In the Node, Host, and Cell Names page, specify the node, host, and cell names for the deployment manager, or accept the defaults and click **Next**. Try to keep the node name as short as possible, but ensure that node names are unique within your deployment environment. See *Naming considerations for profiles, nodes, servers, hosts, and cells* for naming considerations.

   The Administrative Security page displays.

9. Optionally enable administrative security.

   Enable administrative security now by leaving the **Enable administrative security** check box selected, supply a user name and password to log on to the administrative console, and click **Next**.

   The Security Certificate (Part 1) page displays.

10. In the Security Certificate (Part 1) page, create a default personal certificate and a root signing certificate, or import a personal certificate and a root signing certificate from keystore files, and click **Next**.

    You can create both certificates, import both certificates, or create one certificate, and import the other certificate.

    When you import a personal certificate as the default personal certificate, import the root certificate that signed the personal certificate. Otherwise, the Profile Management Tool adds the signer of the personal certificate to the `trust.p12` file. If you import the default personal certificate or the root signing certificate, specify the path and the password, and select the keystore type and the keystore alias for each certificate that you import.

    The Security Certificate (Part 2) page is displayed.

11. In the Security Certificate (Part 2) page, verify that the certificate information is correct, and click **Next**.

    If you create the certificates, you can use the default values or modify them to create new certificates. The default personal certificate is valid for one year by default and is signed by the root signing certificate. The root signing certificate is a self-signed certificate that is valid for 15 years by default. The default keystore password for the root signing certificate is `WebAS`. Change the

password. The password cannot contain any double-byte character set (DBCS) characters because certain keystore types, including PKCS12, do not support these characters. The keystore types that are supported depend on the providers in the `java.security` file.

When you create either or both certificates, or import either or both certificates, the keystore files that are created are:

- `key.p12`: Contains the default personal certificate.
- `trust.p12`: Contains the signer certificate from the default root certificate.
- `root-key.p12`: Contains the root signing certificate.
- `default-signers.p12`: Contains signer certificates that are added to any new keystore file that you create after the server is installed and running. By default, the default root certificate signer and a DataPower signer certificate are in this keystore file.
- `deleted.p12`: Holds certificates deleted with the deleteKeyStore task so that they can be recovered if needed.
- `ltpa.jceks`: Contains server default Lightweight Third-Party Authentication (LTPA) keys that the servers in your environment use to communicate with each other.

These files all have the same password when you create or import the certificates, which is either the default password, or a password that you specify.

An imported certificate is added to the `key.p12` file or the `root-key.p12` file.

If you import any certificates and the certificates do not contain the information that you want, click **Back** to import another certificate.

12. Verify that the ports specified for the profile are unique and click **Next**.

    The Profile Management Tool detects ports currently used by other WebSphere products and displays recommended port values that do not conflict with existing ones. If you have applications other than WebSphere ones that use specified ports, verify that the ports do not conflict. If you chose not to deploy the administrative console on the Optional Application Deployment page in step 6 on page 285, the administrative console ports are not available on the Port Values Assignment page.

    Ports are recognized as being in use if the following conditions are satisfied:
    - They are assigned to a profile created under an installation performed by the current user.
    - They are currently in use.

    Although the tool validates ports when you access the Port Values Assignment page, port conflicts can still occur resulting from selections you make on subsequent Profile Management Tool pages. Ports are not assigned until profile creation completes.

    If you suspect a port conflict, you can investigate it after the profile is created. Determine the ports used during profile creation by examining the following file:

    - Linux UNIX *profile_root*`/properties/portdef.props`
    - Windows *profile_root*`\properties\portdef.props`

    Included in this file are the keys and values used in setting the ports. If you discover port conflicts, you can reassign ports manually. To reassign ports, see the topic Updating ports in an existing profile in the WebSphere Application Server Network Deployment information center. Run the `updatePorts.ant` file through the **ws_ant** script detailed in this topic.

The next step depends on your platform and whether you are installing as a root (Administrator) or nonroot user.

Installation type	Next step
On a Linux or Windows platform, *with root or Administrator group privileges*	The Linux or Windows Service Definition page is displayed. Proceed to step 13.
On any other platform, or as a nonroot user on a Linux or Windows platform	The Database Configuration page is displayed. Proceed to step 14 on page 289.

13. Linux Windows Choose whether to run the process as a Windows service on a Windows platform or as a Linux service on a Linux platform and click **Next**.

    Windows The Windows Service Definition page is displayed for the Windows platform only if the ID that installs the Windows service has the Administrator group privilege. If the profile is configured as a Windows service, the product starts Windows services for processes started by the **startServer** or **startManager** commands. For example, if you configure a server or deployment manager as a Windows service and issue the **startServer** or **startManager** commands, the **wasservice** command starts the defined services.

    Important: If you choose to log on as a specified user account, you must specify the user ID and the password for the user who is to run the service, and the startup type (default is `Manual`). The user ID must not have spaces in its name, it must belong to the Administrator group, and it must have the advanced user right "Log on as a service." If the user ID belongs to the Administrator group, the Profile Management Tool grants it the advanced user right if it does not already have it.

    During profile deletion, you can remove the Windows service that is added during profile creation.

    IPv6 considerations when running profiles as Windows services

    Profiles created to run as a Windows service fail to start when using IPv6 if the service is configured to run as Local System. Create a user-specific environment variable to enable IPv6. Because this environment variable is a user variable instead of a Local System variable, only a Windows service that runs as that specific user can access this environment variable. By default, when a new profile is created and configured to run as a Windows service, the service is set to run as Local System. When the WebSphere ESB Windows service tries to run, the service is unable to access the user environment variable that specifies IPv6, and thus tries to start as IPv4. The server does not start correctly in this case. To resolve the problem, when creating the profile, specify that the WebSphere ESB Windows service runs as the same user ID under which the environment variable that specifies IPv6 is defined, instead of as Local System.

    Linux The Linux Service Definition page is displayed only if the current operating system is a supported version of Linux and the current user has the appropriate permissions.

    WebSphere ESB attempts to start Linux services for processes that are started by the **startServer** or **startManager** commands. For example, if you configure a server or deployment manager as a Linux service and issue the **startServer** or **startManager** commands, the **wasservice** command starts the defined services.

    By default, WebSphere ESB is not selected to run as a Linux service.

To create the service, the user who runs the Profile Management Tool must be the root user. If you run the Profile Management Tool with a non-root user ID, the Linux Service Definition page is not displayed, and no service is created.

You must specify a user name under which the service runs.

To delete a Linux service, the user must be the root user or have proper privileges for deleting the service. Otherwise, a removal script is created that the root user can run to delete the service on behalf of the user.

14. In the Database Configuration page, configure the Common database used by the selected product components.
    a. In the **Choose a database product** field, select the database product you want to use, or accept the default value.

       Restriction: Microsoft SQL Server is not supported on deployment managers using the deployment environment configuration.
    b. To store the database creation and configuration scripts that the profile creation or augmentation process creates in a location other than the default location, select the **Override the destination directory for generated scripts** check box and designate your new location in the **Database script output directory** field. The profile creation or augmentation process creates scripts that you or the database administrator can run manually to create new databases and their required tables, if you choose not to do so during profile creation or augmentation. The process creates scripts for the Common database for all profile types and scripts for the Common Event Infrastructure database for stand-alone server profiles.

       The default locations for the databases are as follows:
       - For the Common Event Infrastructure database:
         - Linux UNIX *install_root*/profiles/*profile name*/dbscripts/CEI_*ceiDbName*
         - Windows *install_root*\profiles*profile name*\dbscripts\CEI_*ceiDbName*
       - For the Common database:
         - Linux UNIX *install_root*/profiles/*profile name*/dbscripts/CommonDB/*dbType*/*dbName*
         - Windows *install_root*\profiles*profile name*\dbscripts\CommonDB*dbType**dbName*

       For selected database products, you can prevent automatic creation and configuration of databases by selecting the **Delay execution of database scripts (must select if using a remote database)**
    c. Enter your Common database name or accept the default value.

       The name of the database on IBM i using Independent Auxiliary Storage Pools (IASPs) can be the name of the IASP.

       Default Common database names differ based on the database product:
       - `*SYSBAS` for DB2 for i5/OS® (Toolbox) and DB2 for iSeries® i (Toolbox)
       - `WPRCSDB` for all other database products

       If you plan to use an existing database, this name must match the name of that database. If you plan to create a new database and the name you specify is already associated with another WebSphere ESB profile, you must use a different database name.

Note: This restriction does not apply to IBM i. All profiles on IBM i use the same database name.

Note: The Oracle database name (dbName) is actually the Oracle Identifier (SID) and must exist in order to create tables. When creating stand-alone server profiles, it can be shared between the Common database and the Common Event Infrastructure database. It is recommended that you remove all Oracle database resources before creating a new profile, because the Common Event Infrastructure database creates unique database resources, such as table spaces, which will fail if they exist in the Oracle server.

d. Select the **Delay execution of database scripts (must select if using a remote database)** check box. A local database will be created if this check box is not selected.

By selecting **Delay execution of database scripts (must select if using a remote database)**, you or the database administrator must manually run the scripts that are stored in the location specified in the **Database script output directory** field on this page.

Restriction: The **Delay execution of database scripts (must select if using a remote database)** option is not available for the following configurations:

- If you chose to create a deployment manager using the Deployment environment option.

For more information about the various databases and database tables the WebSphere ESB product uses, see Planning your database configuration.

See the following topics for instructions on manually creating and configuring databases:

- To create a new Common database or create tables in an existing one, see Creating the Common database and tables after profile creation or augmentation.

Important: Do not use the Common database scripts located in the following directories (where the variable *db_type* represents the supported database product):

- Linux UNIX *install_root*/dbscripts/CommonDB/*db_type*
- Windows *install_root*\dbscripts\CommonDB*db_type*

These default scripts have not been updated by the profile creation or augmentation process.

Restriction: The **Delay execution of database scripts (must select if using a remote database)** option is not available for the following configurations:

- If you chose to create a deployment manager using the Deployment environment option.

e. Click Next to display the Profile Summary page.

15. In the Profile Summary page, click **Create** to create the profile or **Back** to change the characteristics of the profile.
16. Complete the profile configuration by doing one of the following tasks, depending on whether you must manually configure the Common database.
    - Because you decided to postpone database configuration by producing scripts to be run manually, perform the following steps:

a. Clear the check box beside **Launch the First steps console** and click **Finish** to close the Profile Management Tool. Also, close the Profiles page, which is open in a separate window.

b. Use your site's standard database definition tools and procedures to edit and run the scripts the Profile Management Tool generated to create or create and configure the `WPRCSDB` database (or its equivalent if it has a different name on your system).

   The default locations for the database scripts are as follows:

   – For the Common Event Infrastructure database:

     - Linux UNIX *install_root*/profiles/*profile name*/dbscripts/CEI_*ceiDbName*

     - Windows *install_root*\profiles*profile name*\dbscripts\CEI_*ceiDbName*

   – For the Common database:

     - Linux UNIX *install_root*/profiles/*profile name*/dbscripts/CommonDB/*dbType*/*dbName*

     - Windows *install_root*\profiles*profile name*\dbscripts\CommonDB*dbType**dbName*

   Also see *Creating the Common database and tables after profile creation or augmentation* for task descriptions on creating a new Common database or tables in an existing Common database..

Results

You have created a deployment manager profile and you have run the database configuration scripts to create Common database and tables.

What to do next

Start the deployment manager

Start the deployment manager

The deployment manager is a server process. You must start the deployment manager before you can use its administrative console to manage the cell.

Before you begin

You have created the deployment manager profile.

About this task

Perform the following steps to start and stop a deployment manager.

Procedure

1. Start the deployment manager with one of the following actions:
   - Windows From the **Start** menu, select **IBM WebSphere** > **Enterprise Service Bus 7.5** > **Profiles** > *profile_name* > **Start the deployment manager**.
   - In the First steps console, click **Start the deployment manager**.
   - Use the **`startManager`** command.

2. Verify that the deployment manager started successfully by checking the *install_root*/profiles/*profile_name*/logs/*server_name*/startServer.log log file for the message Server *server_name* open for e-business; process id is *nnnn*.

What to do next

Create the custom profiles

Creating custom profiles (managed nodes) for the deployment manager using PMT

Selecting the Advanced option creates profiles with customized configuration settings.

Before you begin

You have created the deployment manager profile and you have run the database configuration scripts to create Common database and tables.

The Deployment manager is running.

About this task

While configuring custom profiles, you can specify your own values for settings such as ports, the location of the profile, and the names for the profile, node, host, and cell (if applicable). You can choose to federate the node to an existing deployment manager during the creation process, or federate it later using the addNode command.

Perform the following steps to create advanced custom profiles (managed nodes).

Procedure

1. Start the WebSphere ESB Profile Management Tool.

   Use one of the following commands:

   - Linux UNIX *install_root*/bin/ProfileManagement/pmt.sh
   - Windows *install_root*\bin\ProfileManagement\pmt.bat

   The Welcome page is displayed.
2. In the Welcome page, click the **Launch Profile Management Tool** button or the **Profile Management Tool** tab.

   The **Profiles** tab is displayed.
3. In the **Profiles** tab, click **Create**.

   The **Profiles** tab can contain a list of profiles that have been created on your machine. For this procedure, it is assumed you are creating a new profile, not augmenting an existing one.

   The Environment Selection page opens in a separate window.
4. In the Environment Selection page, expand WebSphere Enterprise Service Bus and select the type of profile you want to create. Then click **Next**

   The Profile Creation Options page is displayed.
5. In the Profile Creation Options page, choose to perform an **Advanced** profile creation, and click **Next** to display the Optional Application Deployment page.

The **Advanced** option lets you specify your own configuration values for a profile.

By selecting the **Advanced** option, you can do the following:

- Assign customized values to ports, to the location of the profile, and to the names of the profile, node, host, and cell (when applicable).
- Configure the Common database.
- Deploy the administrative console.
- Enable administrative security.
- Create a system service to run the server, if your operating system and the privileges of your user account permit the creation of services.

6. In the Profile Name and Location page, perform the following steps:
   a. Specify a unique name and directory path for the profile, or accept the defaults.

      Each profile that you create must have a name. When you have more than one profile, you can tell them apart at their highest level by this name.

      The directory you specify will contain the files that define the runtime environment, such as commands, configuration files, and log files. The default directory is dependent on platform:

      - Linux UNIX `install_root/profiles/profile_name`
      - Windows `install_root\profiles\profile_name`

      where *profile_name* is the name you specified. An error message is displayed if:

      - The *profile_name* you specify is not unique.
      - The directory you specify is not empty.
      - Your user ID does not have sufficient permissions for the directory.
      - There is insufficient space to create the profile.

   b. You can make the profile you are creating the default profile (so commands work automatically with it) by selecting the **Make this profile the default** check box. This check box appears only if you have an existing profile on your system.

      The first profile that you create on a machine is the default profile.

      The default profile is the default target for commands that are issued from the `bin` directory in the product installation root. When only one profile exists on a machine, every command operates on that profile. If more than one profile exists, certain commands require that you specify the profile to which the command applies. See *Profile commands in a multiprofile environment* for more information.

      The Profile Management Tool detects ports currently used by other WebSphere products, but not the ports of other applications that might use specified ports. When federating a custom profile, the **addNode** command uses non-conflicting ports. This action means that you can take the default port assignments as you create the profile, and let the **addNode** command specify non-conflicting ports as you federate the node. Port assignments must be unique on a server. Server processes on different servers can use the same port assignments without conflict.

   c. Click **Next**. (If you click **Back** and change the name of the profile, you might have to manually change the name on this page when it is displayed again.)

      The Node, Host, and Cell Names page is displayed.

7. In the Node, Host, and Cell Names page, specify the node, host, and cell names for the profile, or accept the defaults and click **Next**. Try to keep the node name as short as possible, but ensure that node names are unique within the deployment environment. See *Naming considerations for profiles, nodes, servers, hosts, and cells* for naming considerations

   The Federation page is displayed.
8. In the Federation page, select the **Federate this node later** check box and click **Next**

   After you create the custom profiles you will federate the nodes using the addNode command.

   How to decide whether or not to federate the node as part of custom profile creation:

   Important:

   Do *not* federate the custom node during profile creation if any one of the following situations is true:
   - You plan to use this custom node as a migration target.
   - Another profile is being federated. (Node federation must be serialized.)
   - The deployment manager is not running or you are not sure if it is running.
   - The deployment manager has not yet been augmented into an WebSphere ESB deployment manager.
   - The deployment manager is not at a release level the same or higher than the release level of the profile you are creating.
   - The deployment manager does not have a JMX administrative port enabled.
   - The deployment manager is re-configured to use the non-default remote method invocation (RMI) as the preferred Java Management Extensions (JMX) connector. (To verify the preferred connector type, in the administrative console of the deployment manager select **System administration** > **Deployment manager** > **Administration services**.)

   Processing associated with federating the node as part of custom profile creation:
   - The Profile Management Tool verifies that the deployment manager exists and can be contacted, and that the authentication user ID and password are valid for that deployment manager (if it is secured).
   - If you attempt to federate a custom node when the deployment manager is not running or is not available for other reasons, a warning box prevents you from continuing. If this warning box appears, click **OK** to exit from it, and then make different selections on the Federation page.

   The Security Certificate (Part 1) page is displayed.
9. In the Security Certificate (Part 1) page, create a default personal certificate and a root signing certificate, or import a personal certificate and a root signing certificate from keystore files, and click **Next**.

   You can create both certificates, import both certificates, or create one certificate, and import the other certificate.

   When you import a personal certificate as the default personal certificate, import the root certificate that signed the personal certificate. Otherwise, the Profile Management Tool adds the signer of the personal certificate to the `trust.p12` file. If you import the default personal certificate or the root signing certificate, specify the path and the password, and select the keystore type and the keystore alias for each certificate that you import.

The Security Certificate (Part 2) page is displayed.

10. Verify that the certificate information is correct, and click **Next**.

    If you create the certificates, you can use the default values or modify them to create new certificates. The default personal certificate is valid for one year by default and is signed by the root signing certificate. The root signing certificate is a self-signed certificate that is valid for 15 years by default. The default keystore password for the root signing certificate is `WebAS`. Change the password. The password cannot contain any double-byte character set (DBCS) characters because certain keystore types, including PKCS12, do not support these characters. The keystore types that are supported depend on the providers in the `java.security` file.

    When you create either or both certificates, or import either or both certificates, the keystore files that are created are:

    - `key.p12`: Contains the default personal certificate.
    - `trust.p12`: Contains the signer certificate from the default root certificate.
    - `root-key.p12`: Contains the root signing certificate.
    - `default-signers.p12`: Contains signer certificates that are added to any new keystore file that you create after the server is installed and running. By default, the default root certificate signer and a DataPower signer certificate are in this keystore file.
    - `deleted.p12`: Holds certificates deleted with the deleteKeyStore task so that they can be recovered if needed.
    - `ltpa.jceks`: Contains server default Lightweight Third-Party Authentication (LTPA) keys that the servers in your environment use to communicate with each other.

    These files all have the same password when you create or import the certificates, which is either the default password, or a password that you specify.

    An imported certificate is added to the `key.p12` file or the `root-key.p12` file.

    If you import any certificates and the certificates do not contain the information that you want, click **Back** to import another certificate.

    The next step depends on whether you elected to federate the profile as part of the profile creation process.

    Because you chose not to federate the node as part of the custom profile creation, the Database Configuration page displays.

11. In the Database Configuration page, perform the following steps:

    a. Review the database product. The database that matches the database used on the deployment manager to which this custom profile will be federated is displayed.

    b. Provide the location (directory) of the JDBC driver class path files for the database. You can accept the default values.

    c. Click **Next**.

    The Profile Summary page is displayed.

12. In the Profile Summary page, click **Create** to create the profile, or **Back** to change the characteristics of the profile.

    When the profile creation is complete, the Profile Complete page is displayed with the message **The Profile Management tool created the profile successfully**.

Attention: If errors are detected during profile creation, other messages might appear in place of the success message, for example:

- **The Profile Management tool created the profile but errors occurred**, which indicates that profile creation completed but errors were generated.
- **The Profile Management tool cannot create the profile**, which indicates that profile creation failed completely.

The Profile Complete page identifies the log file to reference in order to troubleshoot the problems.

13. In the Profile Complete page, clear the checkbox for **Launch the First steps console** and click **Finish** to close the Profile Management Tool. Also, close the Profiles page, which is open in a separate window.

What to do next

Repeat the steps you just completed to create a second custom profile for node 2.

After you have created the second custom profile, you must federate the nodes using the addNode command.

Federate the nodes into the deployment manager shell

You can use the addNode command to federate a custom node into a deployment manager cell.

Before you begin

Before using this procedure, ensure that the following prerequisites are met:

- You have installed WebSphere ESB and created an WebSphere ESB deployment manager and two custom profiles. This procedure assumes you did *not* federate the custom profiles during its creation or augmentation, either with the Profile Management Tool or with the **manageprofiles** command-line utility.
- The deployment manager is running. If it is not, start it either by selecting **Start the deployment manager** from its First steps console or by entering the following command, where *profile_root* represents the installation location of the deployment manager profile:
  - Linux UNIX `profile_root/bin/startManager.sh`
  - Windows `profile_root\bin\startManager.bat`
- The deployment manager has been augmented into an WebSphere ESB deployment manager.
- The deployment manager is at the same release level or higher than the custom profile you created or augmented.
- The deployment manager has a JMX administrative port enabled. The default protocol is SOAP.
- You do not plan to use this custom node as a migration target.

About this task

The following instructions guide you through the process of using the addNode command to federate and deploy custom nodes.

Run this command for each of the custom nodes (profiles) that you created in the previous tasks.

Procedure

1. Go to the `bin` directory of the custom profile you want to federate. Open a command window and go to one of the following directories (from a command line), depending on platform (where *profile_root* represents the installation location of the custom profile):
   - Linux UNIX *profile_root*/bin
   - Windows *profile_root*\bin
2. Issue the **addNode** command.

   Issue one of the following commands from the command line if security is not enabled:
   - Linux UNIX `./addNode.sh` *deployment_manager_host deployment_manager_SOAP_port*
   - Windows `addNode.bat` *deployment_manager_host deployment_manager_SOAP_port*

   Issue one of the following commands from the command line if security is enabled:
   - Linux UNIX `./addNode.sh` *deployment_manager_host deployment_manager_SOAP_port* `-username` *userID_for_authentication* `-password` *password_for_authentication*
   - Windows `addNode.bat` *deployment_manager_host deployment_manager_SOAP_port* `-username` *userID_for_authentication* `-password` *password_for_authentication*

   An output window opens. If you see a message similar to the following message, your custom profile was federated successfully:

   ```
   ADMU0003I: Node DMNDID2Node03 has been successfully federated.
   ```

What to do next

Now you are ready to Create the deployment environment for your ND configuration

Creating deployment environments using the deployment environment wizard

After you select a deployment pattern, use the Deployment Environment Configuration wizard to create the deployment environment that is based on the pattern.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

The procedures for creating deployment environments using the deployment environment wizard includes steps on selecting patterns and features, and therefore it is assumed that you have read and understood the information on patterns and features documented in the Planning section.

It is assumed that you have installed the product and have created the deployment manager profile and the associated nodes.

Additionally, one of the steps in the deployment environment configuration wizard includes importing a database design document. The database design document defines the database configuration for the selected deployment environment features. WebSphere ESB includes a response-driven database design tool (DDT) that prompts users for information on the databases that will be used WebSphere ESB (information such as database components, database types and the topology in which the database will be used). The output of the DDT is a database design document (XML file) that is used by the DDT to create the database scripts. For more information on the DDT and for more information on database configuration in general, see *Configuring databases*.

About this task

This task describes the procedure for creating a deployment environment that is based on a specific pattern and uses the Deployment Environment Configuration wizard.

Note: If you make an error while you are working in the wizard, you can go back by clicking **Back**.

Procedure

1. From the administrative console, go to the Deployment Environments page by clicking **Servers** > **Deployment Environments**
2. Launch the Deployment Environment Configuration wizard by clicking **New** on the Deployment Environments page.
   a. The **Create a deployment environment based on a pattern** option is selected. **Create a deployment environment based on a pattern** is the system default and it is the option described in this topic.

      Deployment environment patterns are rules-based configurations of the most commonly used business integration topologies. A pattern provides you with a template for the deployment environment that you are creating.

      Note: Patterns have a direct relationship to the products supported by the configured deployment manager. WebSphere ESB supports a specific set of patterns, with the *Remote messaging and remote support* pattern being the system default. If your deployment manager supports other products in addition to WebSphere ESB, additional patterns may apply. Consult product-specific documentation for information on patterns as they apply to the products.

      For information on the patterns included with and supported by WebSphere ESB, see *Deployment environment patterns* in the Planning section.

      See *Configuring custom deployment environments* for information on using the Custom Deployment Topology Detail page to configure your custom deployment environment.
   b. Enter a unique name for the deployment environment in the **Deployment environment name** field.
   c. If you want view all of the configuration steps in the wizard, select **Detailed: Show all steps**

If you choose **Fast path: Show only needed steps** the wizard displays only those pages that **do not** have assigned default values. Choose **Fast path: Show only needed steps** only if you are agreeable to accepting the system-provided default values for the deployment environment configuration.

This topic assumes that you have chosen **Detailed: Show all steps**

d. Click **Next** to display the Deployment Environment Features page.

3. On the Deployment Environment Features page, select the feature for the deployment environment and click **Next** to either view a list of *compatible features*, or to view a list of deployment environment patterns.

   Features represent the runtime processing capabilities of your deployment environment.

   The list of available features on the Deployment Environment Features page is based on the deployment manager profile. If your deployment manager profile has been augmented to include other products besides WebSphere ESB (for example, WebSphere Business Monitor or WebSphere Business Services Fabric), then the Deployment Environment Features page will list these features as well.

   If you have only installed and configured a profile for WebSphere ESB, then the Deployment Environment Features page will include the following:

   WESB, for WebSphere Enterprise Service Bus, which provides a deployment environment that supports mediations.

   The default value for the deployment environment feature matches the runtime capabilities of your deployment manager.

4. On the Select compatible deployment environment features page, select additional features as necessary and click **Next** to view the list of patterns associated with your primary and ancillary feature selections.

   Note: The Select compatible deployment environment features page displays only if the deployment manager has been augmented with other business process management (BPM) features, such as WebSphere Business Monitor.

   For an understanding of the relationship of features and compatible features, see the information on deployment environments in the Planing section.

5. On the Select the deployment environment pattern page, select the pattern for this deployment environment, then click **Next** to display the Select Nodes page.

   The list of patterns that display on the Deployment Environment Patterns page is dynamic. This list is activated by, and dependent on, the following environment conditions and configuration decisions:

   - The platform on which you have installed the software
   - The selections that you have made on the Select the deployment environment feature and the Select compatible deployment environment features pages.

   The table below lists all of the available patterns and describes the pattern-to-feature relationship. The table also indicates the default conditions for each pattern.

Table 18. Supplied patterns and their relationship to features

Pattern	Default status	Number of clusters	Description	Supported products and features
Single Cluster	This is the default pattern for a z/OS installation.	1	The single cluster is the simplest pattern that defines one cluster for application deployment. For a detailed description of this pattern, see *Deployment environment patterns* in the Planning section.	Supported by the following single BPM product installations: • WebSphere Business Process Manager • WebSphere ESB • WebSphere Business Services Fabric • WebSphere Business Monitor Supported by the following multi-feature installation: • WebSphere Business Monitor + WebSphere ESB • WebSphere Business Monitor + WebSphere Business Process Manager • WebSphere Business Monitor + WebSphere Business Services Fabric
Remote Messaging	This is not a default pattern.	2	The remote messaging pattern defines one cluster for application deployment and one remote cluster for the messaging infrastructure. For a detailed description of this pattern, see *Deployment environment patterns* in the Planning section.	Supported by the following single BPM product installations: • WebSphere Business Process Manager • WebSphere ESB • WebSphere Business Services Fabric
Remote Messaging and Remote Support	This is the default pattern for the following installations: • WebSphere Business Process Manager on multiplatforms • WebSphere Enterprise Service Bus on multiplatforms • WebSphere Business Services Fabric	3	This pattern defines one cluster for application deployment, one remote cluster for the messaging infrastructure, and one remote cluster for the Common Event Infrastructure and other supporting applications. For a detailed description of this pattern, see *Deployment environment patterns* in the Planning section.	Supported by the following single BPM product installations: • WebSphere Business Process Manager • WebSphere ESB • WebSphere Business Services Fabric

Table 18. Supplied patterns and their relationship to features (continued)

Pattern	Default status	Number of clusters	Description	Supported products and features
Remote Messaging, Support and Web	This is the default pattern for a WebSphere Business Monitor installation	4	This pattern defines one cluster for application deployment, one remote cluster for the messaging infrastructure, one remote cluster for supporting applications and one for web applications. For a detailed description of this pattern, see *Deployment environment patterns* in the Planning section.	Supported by the following single BPM product installations: • WebSphere Business Monitor Supported by the following multi-feature BPM installation: • WebSphere Business Monitor + WebSphere ESB • WebSphere Business Monitor + WebSphere Business Process Manager • WebSphere Business Monitor + WebSphere Business Services Fabric

6. Optional: On the Select Nodes page, select the nodes to include in this deployment environment, then click **Next** to display the Cluster members page.

   To include a node, select the check box next to the node name. Use **Node Mapping** to map the selected node to another node name.

7. Optional: On the Clusters page, assign the required number of cluster members on each node for each cluster *type* (Application Deployment Target, Messaging Infrastructure and Supporting Infrastructure) of the deployment environment.

   By default one cluster member is assigned on each node for each function. You change the number by replacing the number in each column. If you are unfamiliar with the different cluster roles and functions provided by each type of cluster, see "Topology types and deployment environment patterns."

   A 0 (zero) value for a node means that the node does not contribute to the selected function, based on features that you have selected.

   After assigning cluster members, you can click **Next** to display the Cluster naming pages for each cluster type of the deployment environment. The Cluster naming sub-steps that display will vary depending on the deployment environment pattern selected.

   The system generates default values for cluster names and cluster member names.

   If you do not want to customize cluster names or cluster member names, you can use the wizard navigation pane to go directly to the REST Services page.

   a. Optional: Customize the cluster names and cluster member names.

      Use the Cluster Naming page to customize cluster names or cluster member names for the cluster type. There is one substep page for each cluster *type* in the pattern that you have selected. For example, if you selected a **Remote messaging and remote support pattern**, there are 3 sub-steps, one for each type of cluster (Application Deployment Target, Messaging Infrastructure and Supporting Infrastructure) in that pattern.

The information on each substep page is as follows:

Cluster
A read-only field specifying the functional role of the cluster.

The value varies depending on the cluster type, as follows:

- Application Deployment Target
- Supporting Infrastructure
- Messaging Infrastructure

For information on the functional role provided by each cluster type, see Topologies and deployment environment patterns in the Planning section.

Cluster Name
Contains the system-generated default value for the cluster name.

The default values are based on a naming convention of `<Deployment Environment Name>.<Cluster type name>`, where`cluster type name` is one of the following values:

- AppTarget

  For clusters performing the role of application deployment target
- Messaging

  For clusters performing the role of messaging infrastructure
- Support

  For clusters performing the role of supporting infrastructure
- Web

  For clusters performing the role of supporting web applications.

  Note: This cluster type name applies for BPM configurations in which WebSphere Business Monitor is the primary feature / product.

Cluster Member Name
Accept the system-generated default value or specify a name of your choosing.

The default value for the cluster member name is based on the following naming convention: `<cluster name>.<node name>.<node number sequence>` .

The number of cluster member names that display in the table match the number of cluster members that you entered for the cluster type column and node row on the Clusters page. See the preceding step for the Clusters page.

8. On the REST Services page, configure service endpoints for Representational State Transfer (REST) application programming interfaces (APIs).

   If you want widgets to be available in Business Space, you must configure the REST service endpoints for those widgets.

   a. Configure a full URL path for all REST services by selecting either **https://** or **http://** from the **Protocol** list.
   b. Enter a name in the **Host Name or Virtual Host in a Load-Balanced Environment**
   c. Enter the **Port** that a client needs to communicate with the server or cluster.

d. In the table of REST services, if you want to modify the description of the REST service endpoint, type in the Description field. The other fields are read-only.

e. Click **Next** to go to the Import the database configuration page.

9. Optional: If you have a database design document, you can use this page to import it. Import the database configuration by selecting **Browse** to go the database design XML document or enter the path to the database design XML document and then click **Next** to go to the Data sources page.

   The design document can be based on a database design that you created using the database design tool (DDT), or it can be the supplied design document based on the pattern and feature that you have selected.

   If you do not have a database design document, you can use the Database page to create your database configuration.

10. Conditional optional: On the Database page, configure the database parameters for data sources of the deployment environment, then click **Next** to go to the Security page.

    On this page, define the database information for the components that are included in this deployment environment. Where possible, the wizard supplies default information for the parameters, but change those values to match the values that you defined when you planned the environment.

    Note: If you imported a database design document, the information on the Database page reflects the data source configuration as defined in the database design document that you imported in step 9.

    Whether or not this step displays for a fast path deployment environment configuration is conditional. This step displays for a fast path deployment environment configuration if more than one database has been defined.

    This step always displays if you are using DB2 for z/OS or an Oracle database provider.

    The default schema names that display on this page may conflict with your site naming convention or may even collide with existing schemas, and as such it is likely that you will need to change the schema name.

    Oracle database considerations:

    - If you do not want to provide a DBA user name and password for all components when using Oracle, clear **Create tables** and specify preexisting and unique user names and passwords for each component. If you are able to provide a DBA user name and password for all the components, select **Create tables** and allow the configuration process to create the required schemas and users.

    For a production environment, you should set the same values for **User name** and **Schema name** and you should deselect **Create tables**. For a production environment, create the required schemas manually and use the SQL files generated to create the tables.

    Note: You cannot select **Create tables** for Business Space (the option will be "greyed out"). You need to run the SQL files for Business Space manually. For information on running the SQL manually for Business Space, see *Configuring Business Space database tables.*

    You can edit all key parameters, such as database name; whether or not to create tables, the data source runtime user name, password for the deployment environment.

    You can select which database to use for the given component.

DB2 for z/OS: The **Create tables** option cannot be used if you are using a DB2 for z/OS database provider.

Steps that cannot be completed through the Deployment Environment Configuration wizard, and which need to be completed manually, are listed on the Deferred Configuration page.

11. On the Security page, configure the authentication aliases WebSphere uses when accessing secure components

    The authentication alias user name and password can be changed on this page. These aliases are used to access secure components but do not provide access to data sources

12. Optional: On the System web applications page, set the context root for component-based web applications in your deployment environment or accept the system-provided default values for the context roots. Then click **Next** to display the Summary page.

    The table contains the following control information.

    Web Application
    The name of the Web application.

    Some of the components that are part of the deployment environment you are creating contain web applications. The **Web application** column can include the following components:

    - Business Space

    Context Root
    The current value of the context root for the component.

    By default, the default context root for the web application applies. You can change the context roots by typing over the value in the **Context Root** field.

    Note: The Business Space context root is read only and cannot be edited.

13. Verify that the information on the Summary page is correct and click **Finish and Generate Environment** to save and complete the configuration of the deployment environment. To exit without completing the configuration, click **Finish**.

    Clicking **Finish** saves the deployment environment configuration - but does not generate it.

    Click **Cancel** cancels the deployment configuration and does not save the configuration.

    a. Check for deferred configuration steps

       Select **Deployment Environments** > *name of deployment environment* > **Deferred Configuration**

       You need to address any existing deferred configuration steps before starting the Deployment Environment.

Results

When the configuration completes, you can examine the configuration files to view the changes.

What to do next

If you have deferred configuration steps, address them and complete the configuration.

Tutorials about administering the server in Business Space

Tutorial: Creating mediation policies for modules, using Business Space

You can use mediation policies to control message flows between service requesters and service providers. Mediation policies can be associated either with modules or with target services: this tutorial deals with mediation policies associated with modules. This tutorial uses Business Space widgets to create mediation policies.

This tutorial shows you how to load an SCA module into WebSphere Service Registry and Repository (WSRR). Then the tutorial shows you how to use Business Space widgets to create mediation policies for your SCA module.

Note: You can use Business Space widgets to create mediation policies and conditions. However, you cannot use Business Space widgets to govern your policies.

The tutorial is divided into chunks of information called education modules, and each education module has one or more lessons.

Introduction

If you associate mediation policies with suitable SCA modules, you can control the dynamic properties of mediations.

When you use WebSphere Integration Developer to create SCA modules containing mediation flows, any module property that you promote (make visible to the run time) is also a dynamic property. Dynamic properties can be overridden, at run time, using mediation policies in a registry.

Learning objectives

- Understand how SCA modules and mediation policies relate to each other.
- Learn how to load SCA modules into WSRR.
- Learn how to create a mediation policy.
- Learn how to create a conditional mediation policy.

This tutorial takes approximately 30 minutes to complete. If you explore other concepts related to this tutorial, it could take longer to finish.

Skill level

Intermediate.

Audience

Integration Developers.

Administrators.

System requirements

- WebSphere Integration Developer, Version 7.0
- WebSphere Enterprise Service Bus, Version 7 or WebSphere Process Server, Version 7.0
- WebSphere Service Registry and Repository, Version 7.0

Note: You can use WebSphere Service Registry and Repository (WSRR), Version 6.2 or above, but the instructions in this tutorial are based on WSRR Version 7.0

Prerequisites

1. Read the mediation policies scenario, which explains the overview. The mediation policies scenario also explains how the developer, the runtime administrator, and the WSRR administrator work together.
2. In WebSphere Integration Developer, create and configure an SCA module containing a Policy Resolution mediation primitive.
3. If you have not yet configured your runtime environment, do so now. For example, from the administrative console deploy your SCA module and ensure that you have a suitable WSRR definition for your SCA module.

Loading an SCA module into WSRR

A mediation policy can change the dynamic properties of a mediation module, or any SCA module that has a mediation flow component. This education module shows you how to load an SCA module into WSRR, so that you can attach a mediation policy to it.

Learn how to load an SCA module into WSRR.

5 minutes.

Loading an SCA module:

Load an SCA module, in an EAR file, from the local file system.

If you have a versioned SCA module to deploy to clusters, run **`CreateVersionedSCAModule`**. **`CreateVersionedSCAModule`** creates an EAR file that contains a unique instance of a versioned SCA module. Create one instance of the module for every cluster that you want to deploy to. Then load your versioned SCA modules into WSRR from the new EAR files.

When you load the EAR file containing your SCA module, WSRR creates an SCA module document. WSRR also loads any system-generated mediation policies.

1. Log on to the WSRR Web user interface. To start the Web user interface, enter the following URL into the address bar of the Web browser: `http://hostname:port/ServiceRegistry` where `hostname` is the IP address or host name of the WebSphere Application Server that is hosting the WSRR application, and `port` is the server port number.
2. If necessary, switch to the WSRR Administrator perspective. Click **Perspective** > **Administrator**. This tutorial assumes that you are using the Administrator perspective.
3. Click **View** > **Service Documents** > **SCA Integration Modules**. The SCA Integration Modules page is displayed.
4. Click **Load Documents**. The Load Documents page is displayed.
5. Click **Browse**, to find the SCA integration module that you want to load.
6. Select the EAR file containing the SCA module that you want to load. The path to the EAR file is displayed.
7. Enter the document description.
8. Optional: Enter the version. The version referred to on this panel is the WSRR document version, not the SCA module version (every WSRR document can have a version identifier). If you load a versioned SCA module, the runtime

registry query needs to find the correctly versioned SCA module. The WSRR document version is not relevant, in this context. If you want, you can keep the SCA module version the same as the WSRR document version. To keep the version identifiers the same you must manually enter the SCA module number. For example, if your EAR file is called CreditHistory_v1_0_0App.ear and your SCA module is called CreditHistory_v1_0_0 enter the version number: v1_0_0.

Note: You cannot change the WSRR version identifier after you have loaded your SCA module.

9. Click **OK**.

WSRR creates an SCA module document.

Example: Loading an SCA integration module

Starting and configuring Business Space

Configure Business Space to administer mediation policies. This education module shows you how to log on to Business Space and configure the Business Space environment.

Learn how to log on to Business Space and create a business space containing the Module Browser widget and the Mediation Policy Administration widget.

5 minutes.

Starting Business Space:

Start Business Space and log on.

1. Open a Web browser.
2. Type the Business Space URL. The Business Space URL is: `http://yourhost:yourport/BusinessSpace` where *yourhost* is the IP address of your server, and *yourport* is the port of your server, for the HTTP transport. For example, the URL might be http://localhost:9080/BusinessSpace. The login screen is displayed.
3. If security is enabled, enter a valid user ID and password. The default user ID is admin.
4. Click **Login**.

A Business Space start screen is displayed. The start screen can vary, depending on how you have configured it.

Example: Business Space Login

The following figure shows the Business Space Login screen, if security is enabled.

Figure 116. Business Space Login

Configuring Business Space for mediation policies:

Create a business space that includes the Module Browser widget and the Mediation Policy Administration widget.

When you configure the Business Space tool, you create your own business spaces. For this tutorial, you need a business space that contains the Module Browser widget and the Mediation Policy Administration widget.

1. Click **Actions** > **Create Space**. The Create Space window is displayed.
2. Type a name for the new business space.
3. Optional: Type a description for the new business space.
4. Leave **Create a new space using a template** to the default value.
5. Optional: Select a **Space style**.
6. Click **Save**.
7. Click the business space name.
8. Click the drop-down menu for the page.
9. Click **Edit Settings**
10. Enter the **Page name**.
11. Click **Save**.
12. Click **Edit Page**.
13. Drag a **Module Browser** widget from the palette to the page.
14. Drag a **Mediation Policy Administration** widget from the palette to the page.
15. Click the **Save this page** icon.
16. Click **Finish Editing**.

After the browser has been refreshed, your new business space is displayed. The business space has a page containing the Module Browser widget and the Mediation Policy Administration widget.

Example: A business space for modules and mediation policies

The following figure shows a page, called Modules and Mediation Policies, that contains a Module Browser widget and the Mediation Policy Administration widget.

Figure 117. Administering mediation policies associated with modules, using the Mediation Policy Administration widget

Creating a mediation policy for an SCA module

If you want to override the dynamic properties of an SCA module, you can create mediation policies. This education module shows you how to create a mediation policy and how to create gate conditions that restrict its use.

Learn how to create a mediation policy, using Business Space widgets.

15 minutes.

Creating a mediation policy:

Create a mediation policy for your SCA module.

1. Log on to Business Space.
2. Create a business space that contains the Module Browser widget and the Mediation Policy Administration widget.

If you want to override specific dynamic properties of an SCA module, you can create a mediation policy and define assertions. Each assertion can override one dynamic property.

1. In the Module Browser widget, click the toggle icon beside the module that you want to work with. You can now view some of the module details.
2. In the expanded view, click **Mediation Policies**. The Mediation Policy Administration widget is refreshed. If the module you chose exists on the selected WSRR, and has policy attachments, the policy attachments are displayed.
3. If you have more than one WSRR definition, select the WSRR where you loaded your SCA module. If there are mediation policies attached to the SCA module they are displayed. If the SCA module you are working with is not loaded on the selected WSRR, a message is displayed.
4. Enter the name of the **New policy attachment**. Mediation policy attachments associate a mediation policy with a module. In WSRR, the mediation policy and policy attachment are separate objects.
5. Click **Create**. The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of the new mediation policy.
6. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
7. Enter a name in the **New policy** field. This name is used for the mediation policy you want to create and attach to the module.
8. Click **Next**. The Mediation Policy Administration widget is refreshed. You can now add assertions and gate conditions.

   Note: Using Business Space, you cannot edit assertions after you create a mediation policy. Therefore, you must add all the assertions that you require before you save the mediation policy.
9. Create one or more assertions. Assertions are module properties that the policy can override. In WSRR, the module properties that you want to override appear as policy assertions.

   Note: The widget requires each policy attachment to have at least one assertion.

   a. Select a **Property name**. The name is the alias name of the property. The alias name identifies the property in the mediation flow.

b. Enter a suitable value in the **Value** field; for example, `All` or `10` or `/body/input/address`. When available, the policy value takes precedence at run time. If a policy is not found, or is unsuitable, the run time uses the promoted property value.
c. Click **Add Assertion**.

10. Optional: If you want to delete an assertion, click the delete icon of the appropriate assertion. If you hover over an assertion, the delete icon, a cross, appears at the end of the row.
11. Click **Save**.

In WSRR, a mediation policy and a policy attachment are created. The policy attachment connects the mediation policy to the module. When the creation process is complete, the widget refreshes and displays the new policy attachment in the table.

Example: Creating assertions

The following figure shows two assertions that have been defined, and one assertion that is in the process of being defined.

Figure 118. Adding assertions, using the Mediation Policy Administration widget

Creating a mediation policy with conditions:

Create a mediation policy with gate conditions that restrict its use. At run time, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated policy can be used.

1. In WebSphere Integration Developer, create conditions on the Policy Resolution mediation primitive. Create a **`Policy condition name`** that matches the gate condition on the mediation policy attachment. Specify where the condition value is found in the message, by providing an XPath expression. For example, you could create a **`Policy condition name`** of Age and an XPath expression of `/body/priceChecker/request/Age`. At run time, the value of condition Age would be set from `/body/priceChecker/request/Age`.
2. Log on to Business Space.
3. Create a business space that contains the Module Browser widget and the Mediation Policy Administration widget.

If you want to override specific dynamic properties of an SCA module, you can create a mediation policy and define assertions. Each assertion can override one dynamic property.

If you want certain conditions to be met by a message, before a mediation policy can be used, define gate conditions.

1. In the Module Browser widget, click the toggle icon beside the SCA module that you want to work with. You can now view some of the module details.
2. In the expanded view, click **Mediation Policies**. The Mediation Policy Administration widget is refreshed. If the module you chose exists on the selected WSRR, and has policy attachments, the policy attachments are displayed.
3. If you have more than one WSRR definition, select the WSRR where you loaded your SCA module. If there are mediation policies attached to the SCA module they are displayed. If the SCA module you are working with is not loaded on the selected WSRR, a message is displayed.
4. Enter the name of the **New policy attachment**. Mediation policy attachments associate a mediation policy with a module. In WSRR, the mediation policy and policy attachment are separate objects.
5. Click **Create**. The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of the new mediation policy.
6. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
7. Enter a name in the **New policy** field. This name is used for the mediation policy that you want to create and attach to the module.
8. Click **Next**. The Mediation Policy Administration widget is refreshed. You can now add assertions and gate conditions.

   Note: Using Business Space, you cannot edit assertions after you create a mediation policy. Therefore, you must add all the assertions that you require before you save the mediation policy.
9. Create one or more assertions. Assertions are module properties that the policy can override. In WSRR, the module properties that you want to override appear as policy assertions.

   Note: The widget requires each policy attachment to have at least one assertion.

   a. Select a **Property name**. The name is the alias name of the property. The alias name identifies the property in the mediation flow.
   b. Enter a suitable value in the **Value** field; for example, `All` or `10` or `/body/input/address`. When available, the policy value takes precedence at run time. If a policy is not found, or is unsuitable, the run time uses the promoted property value.
   c. Click **Add Assertion**.
10. Create one or more gate conditions. Gate conditions must be met before the policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.

    a. Enter a gate condition name in the **Gate condition name** field. The name of a gate condition is always prefixed with the string `medGate_`.
    b. Enter a gate condition value in the **Value** field. The gate condition value is made up of the following parts: *policy condition name*, *operation* and *gate value*.
       - The *policy condition name* you enter must map to a **`Policy condition name`** in the module.
       - The *operation* can be: `=` , `!=` , `>` , `<` , `<=` or `>=` .

- The *gate value* is the value being compared, for example, `country = France`.

c. Click **Add Gate Condition**.

11. Optional: If you want to delete an assertion or gate condition, click the delete icon of the appropriate assertion or gate condition. If you hover over an assertion or gate condition, the delete icon, a cross, appears at the end of the row.
12. Click **Save**.

In WSRR, a mediation policy and a policy attachment are created. The policy attachment connects the mediation policy to the module.

For every gate condition you defined, WSRR creates a user property on the policy attachment. The user property represents the condition. In the mediation policy processing model, mediation policies with gate conditions have the highest precedence. Therefore, assertions defined by these mediation policies have the highest precedence. Mediation policies without gate conditions have a lower precedence. Therefore, assertions defin

ed by these mediation policies have a lower precedence. Administrative console values are used if there is no suitable property in a mediation policy.

When the creation process is complete, the widget refreshes, and displays the new policy attachment in the table.

Example: Creating assertions and conditions

The following figure shows two assertions and one gate condition that have been defined, and one gate condition that is in the process of being defined.

Figure 119. Adding assertions and gate conditions, using the Mediation Policy Administration widget

Attaching an existing mediation policy to an SCA module

Attach existing mediation policies to modules. At the same time, create gate conditions that restrict the use of the mediation policy.

Learn how to reuse existing mediation policies.

5 minutes.

Attaching an existing mediation policy:

You can reuse mediation policies by attaching them to different modules. When you attach an existing mediation policy to a module, you can create gate conditions that restrict its use.

1. In WebSphere Integration Developer, create conditions on the Policy Resolution mediation primitive. Create a **`Policy condition name`** that matches the gate condition on the mediation policy attachment. Specify where the condition value is found in the message, by providing an XPath expression. For example, you could create a **`Policy condition name`** of Age and an XPath expression of `/body/priceChecker/request/Age`. At run time, the value of condition Age would be set from `/body/priceChecker/request/Age`.
2. Log on to Business Space.
3. Create a business space that contains the Module Browser widget and the Mediation Policy Administration widget.

You might decide to create reusable mediation policies. For example, you might have a mediation policy that turns on logging. All modules that wanted to use the mediation policy for logging would define their logging property using the same alias name. Then the modules could override the logging value using the mediation policy. Different modules might want to enable logging under different conditions. For example, one module might want to log a service request if a bank withdrawal was greater than a certain amount. Different modules can use the same mediation policy under different conditions, by creating gate conditions on the policy attachment.

At run time, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated policy can be used.

1. In the Module Browser widget, click the toggle icon beside the SCA module that you want to work with. You can now view some of the module details.
2. In the expanded view, click **Mediation Policies**. The Mediation Policy Administration widget is refreshed. If the module you chose exists on the selected WSRR, and has policy attachments, the policy attachments are displayed.
3. If you have more than one WSRR definition, select the WSRR where you loaded your SCA module. If there are mediation policies attached to the SCA module they are displayed. If the SCA module you are working with is not loaded on the selected WSRR, a message is displayed.
4. Enter the name of the **New policy attachment**. Mediation policy attachments associate a mediation policy with a module. In WSRR, the mediation policy and policy attachment are separate objects.
5. Click **Create**. The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of the new mediation policy.
6. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
7. Click **Use existing**.
8. Select a mediation policy from the **Select a policy** menu. The mediation policies displayed depend on the group you selected.
9. Click **Next**. The Mediation Policy Administration widget is refreshed. You can now add gate conditions.

Note: Using Business Space, you cannot edit mediation policy assertions after you create a mediation policy. However, because gate conditions exist on the mediation policy attachment, you can add gate conditions when you create a policy attachment.

10. Create one or more gate conditions. Gate conditions must be met before the policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.
    a. Enter a gate condition name in the **Gate condition name** field. The name of a gate condition is always prefixed with the string `medGate_`.
    b. Enter a gate condition value in the **Value** field. The gate condition value is made up of the following parts: *policy condition name*, *operation* and *gate value*.
       - The *policy condition name* you enter must map to a **`Policy condition name`** in the module.
       - The *operation* can be: `=` , `!=` , `>` , `<` , `<=` or `>=` .
       - The *gate value* is the value being compared, for example, `country = France`.
    c. Click **Add Gate Condition**.
11. Optional: If you want to delete a gate condition, click the delete icon of the appropriate gate condition. If you hover over a gate condition, the delete icon, a cross, appears at the end of the row.
12. Click **Save**.

In WSRR, a policy attachment is created. The policy attachment connects the mediation policy to the module.

For every gate condition you defined, WSRR creates a user property on the policy attachment. The user property represents the condition. In the mediation policy processing model, mediation policies with gate conditions have the highest precedence. Therefore, assertions defined by these mediation policies have the highest precedence. Mediation policies without gate conditions have a lower precedence. Therefore, assertions defined by these mediation policies have a lower precedence. Administrative console values are used if there is no suitable property in a mediation policy.

When the creation process is complete, the widget refreshes and displays the new policy attachment in the table.

Example: Selecting an existing mediation policy

The following figure shows the mediation policies that exist for the group DomainB. You can select one of these mediation policies and attach it to your module.

Figure 120. Selecting an existing mediation policy, to attach to a module

Tutorial summary

This mediation policy tutorial shows you how to use Business Space widgets to create and reuse mediation policies associated with modules. In addition, this tutorial shows you how to use Business Space widgets to create conditions that restrict the use of mediation policies.

After you have completed this tutorial, you can run the application associated with your SCA module. Depending on the message being processed, and on any gate conditions, the mediation flow can be altered by mediation policies.

Tutorial: Creating mediation policies for services, using Business Space

You can use mediation policies to control message flows between service requesters and service providers. Mediation policies can be associated either with modules or with target services. This tutorial uses Business Space widgets to create mediation policies associated with target services.

You can associate mediation policies at different levels of the target service. The levels supported by the Business Space widgets are service, endpoint, and operation. This tutorial shows you how to use Business Space widgets to create mediation policies for WSDL-defined services, endpoints, and operations.

Note: You can use Business Space widgets to create mediation policies and conditions. However, you cannot use Business Space widgets to govern your policies.

The tutorial is divided into chunks of information called education modules, and each education module has one or more lessons.

Introduction

If you associate mediation policies with target services, you can override the dynamic properties of mediations before the target services are called.

When you use WebSphere Integration Developer to create SCA modules containing mediation flows, any module property that you promote (make visible to the run time) is also a dynamic property. Dynamic properties can be overridden, at run time, using mediation policies in a registry.

The WSDL documents that define your services must exist in WSRR, because the WSDL defines the levels at which you can attach mediation policies. The supported levels are: service, endpoint, and operation.

Note:

- A WSDL endpoint can contain a port, a binding, and a portType. When you use Business Space to attach a mediation policy to the endpoint level, the mediation policy is attached to the endpoint portType.
- In WSRR, the operation object has two associated objects: portType operation objects and binding operation objects. When you use Business Space to attach a mediation policy to the operation level, the mediation policy is attached to the operation portType.

Learning objectives

- Understand how SCA modules, target services, and mediation policies relate to each other.
- Learn how to create a mediation policy associated with a target service.
- Learn how to create a conditional mediation policy associated with a target service.

This tutorial takes approximately 25 minutes to complete. If you explore other concepts related to this tutorial, it could take longer to finish.

Skill level

Intermediate.

Audience

Integration Developers.

Administrators.

System requirements

- WebSphere Integration Developer, Version 7.0
- WebSphere Enterprise Service Bus, Version 7 or WebSphere Process Server, Version 7.0
- WebSphere Service Registry and Repository, Version 7.0

  Note: You can use WebSphere Service Registry and Repository (WSRR), Version 6.2 or above, but the instructions in this tutorial are based on WSRR Version 7.0

Prerequisites

1. Read the mediation policies scenario, which explains the overview. The mediation policies scenario also explains how the developer, the runtime administrator, and the WSRR administrator work together.
2. In WebSphere Integration Developer, create and configure an SCA module containing a Policy Resolution mediation primitive and an Endpoint Lookup mediation primitive.
3. In WSRR, load the enterprise archive (EAR) file containing your module. Also, load the WSDL documents for the services that you want to attach mediation policies to.
4. If you have not yet configured your runtime environment, do so now. For example, from the administrative console deploy your SCA module and ensure that you have a suitable WSRR definition.

Starting and configuring Business Space

Configure Business Space to administer mediation policies. This education module shows you how to log on to Business Space and configure the Business Space environment.

Learn how to log on to Business Space and create a business space containing the Service Browser widget and the Mediation Policy Administration widget.

5 minutes.

Starting Business Space:

Start Business Space and log on.

1. Open a Web browser.
2. Type the Business Space URL. The Business Space URL is: `http://yourhost:yourport/BusinessSpace` where *yourhost* is the IP address of your server, and *yourport* is the port of your server, for the HTTP transport. For example, the URL might be http://localhost:9080/BusinessSpace. The login screen is displayed.
3. If security is enabled, enter a valid user ID and password. The default user ID is admin.
4. Click **Login**.

A Business Space start screen is displayed. The start screen can vary, depending on how you have configured it.

Example: Business Space Login

The following figure shows the Business Space Login screen, if security is enabled.

Figure 121. Business Space Login

Configuring Business Space for services and mediation policies:

Create a business space that includes the Service Browser widget and the Mediation Policy Administration widget.

When you configure the Business Space tool, you create your own business spaces. For this tutorial, you need a business space that contains the Service Browser widget and the Mediation Policy Administration widget.

1. Click **Actions** > **Create Space**. The Create Space window is displayed.
2. Type a name for the new business space.
3. Optional: Type a description for the new business space.
4. Leave **Create a new space using a template** to the default value.
5. Optional: Select a **Space style**.
6. Click **Save**.
7. Click the business space name.
8. Click the drop-down menu for the page.
9. Click **Edit Settings**.
10. Enter the name of the page in the **Page name** field.
11. Click **Save**.
12. Click **Edit Page**.

13. Drag a **Service Browser** widget from the palette to the page.
14. Drag a **Mediation Policy Administration** widget from the palette to the page.
15. Click the **Save this page** icon.
16. Click **Finish Editing**.

After the browser has been refreshed, your new business space is displayed. The business space has a page containing the Service Browser widget and the Mediation Policy Administration widget.

Example: A business space for services and mediation policies

The following figure shows a page, called Services and Mediation Policies, that contains a Service Browser widget and a Mediation Policy Administration widget.

Figure 122. Administering mediation policies associated with services, using the Mediation Policy Administration widget

Creating a mediation policy associated with a service

If you want to override dynamic properties before you call certain services, you can create mediation policies associated with target services. This education module shows you how to create a mediation policy, associated with a target service, and how to create gate conditions that restrict its use.

Learn how to create a mediation policy, using Business Space widgets.

15 minutes.

Creating a mediation policy:

Create a mediation policy for your target service.

1. Log on to Business Space.

2. Create a business space that contains the Service Browser widget and the Mediation Policy Administration widget.

If you want to override dynamic properties before you call certain services, you can create a mediation policy that is associated with a WSDL-defined service, endpoint, or operation. For each dynamic property that you want to override you define one assertion in the mediation policy.

1. From the Service Browser widget, if the correct WSRR definition is not displayed select the correct WSRR definition. If your application server has definitions for more than one instance of WSRR, you can display the services that are defined on each WSRR. The list of services is refreshed.
2. Select the level at which you want to create a mediation policy. You can attach a mediation policy at the level of the service, endpoint, or operation. The Mediation Policy Administration widget is refreshed. The following information is displayed:
   - The name of the service, endpoint, or operation that you selected.
   - The WSRR definition that you selected.
   - Any policy attachments that exist for the service, endpoint, or operation that you selected.
3. Enter the name of the mediation policy attachment in the **New policy attachment** field. Policy attachments associate a mediation policy with a target service. In WSRR, the mediation policy and the policy attachment are separate objects.
4. Click **Create**. The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of the new mediation policy.
5. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
6. Enter a name in the **New policy** field. This is the name of the mediation policy you want to create and attach to the service, endpoint, or operation.
7. Click **Next**. The Mediation Policy Administration widget is refreshed. You can now add assertions and gate conditions.

   Note: Using Business Space, you cannot edit assertions after you create a mediation policy. Therefore, you must add all the assertions that you require before you save the mediation policy.
8. Define one or more assertions. Assertions are module properties that the mediation policy can override. In WSRR, the module properties that you want to override appear as policy assertions.

   Note: The widget requires each policy attachment to have at least one assertion.
   a. Select a **Property name**. The name is the alias name of the property. The alias name identifies the property in the mediation flow.
   b. Enter a suitable value in the **Value** field; for example, `All` or `10` or `/body/input/address`. When available, the policy value takes precedence at run time. If a policy is not found, or is unsuitable, the run time uses the promoted property value.
   c. Click **Add Assertion**.
9. Optional: If you want to delete an assertion, click the delete icon of the appropriate assertion. If you hover over an assertion, the delete icon, a cross, appears at the end of the row.

10. Click **Save**.

In WSRR, a mediation policy and a policy attachment are created. The policy attachment connects the mediation policy to the service, endpoint, or operation. When the creation process is complete, the widget refreshes and displays the new policy attachment in the table.

Example: Defining a new mediation policy

The following figure shows the selection of a group name, called DomainA, from which to create a mediation policy, and the entry of a new mediation policy name.

Figure 123. Selecting the group from which to create a mediation policy, using the Mediation Policy Administration widget

Creating a mediation policy with conditions:

Create a mediation policy for your target service, with gate conditions that restrict its use. At run time, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated policy can be used.

1. In WebSphere Integration Developer, create conditions on the Policy Resolution mediation primitive. Create a **Policy condition name** that matches the gate condition on the mediation policy attachment. Specify where the condition value is found in the message, by providing an XPath expression. For example, you could create a **Policy condition name** of Age and an XPath expression of `/body/priceChecker/request/Age`. At run time, the value of condition Age would be set from `/body/priceChecker/request/Age`.
2. Log on to Business Space.
3. Create a business space that contains the Service Browser widget and the Mediation Policy Administration widget.

If you want to override specific dynamic properties before you call certain services, you can create a mediation policy that is associated with a WSDL-defined service, endpoint, or operation. For each dynamic property that you want to override you define one assertion in the mediation policy.

If you want certain conditions to be met by a message, before a mediation policy can be used, define gate conditions.

1. Log on to Business Space.
2. In your Service Administration business space, open a Service Administration page.
3. From the Service Browser widget, if the correct WSRR definition is not displayed select the correct WSRR definition. If your application server has definitions for more than one instance of WSRR, you can display the services that are defined on each WSRR. The list of services is refreshed.
4. Select the level at which you want to create a mediation policy. You can attach a mediation policy at the level of the service, endpoint, or operation. The Mediation Policy Administration widget is refreshed. The following information is displayed:
   - The name of the service, endpoint, or operation that you selected.
   - The WSRR definition that you selected.
   - Any policy attachments that exist for the service, endpoint, or operation that you selected.
5. Enter the name of the mediation policy attachment in the **New policy attachment** field. Policy attachments associate a mediation policy with a target service. In WSRR, the mediation policy and the policy attachment are separate objects.
6. Click **Create**. The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of the new mediation policy.
7. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
8. Enter a name in the **New policy** field. This is the name of the mediation policy you want to create and attach to the service, endpoint, or operation.
9. Click **Next**. The Mediation Policy Administration widget is refreshed. You can now add assertions and gate conditions.

   Note: Using Business Space, you cannot edit assertions after you create a mediation policy. Therefore, you must add all the assertions that you require before you save the mediation policy.

10. Define one or more assertions. Assertions are module properties that the mediation policy can override. In WSRR, the module properties that you want to override appear as policy assertions.

    Note: The widget requires each policy attachment to have at least one assertion.

    a. Select a **Property name**. The name is the alias name of the property. The alias name identifies the property in the mediation flow.
    b. Enter a suitable value in the **Value** field; for example, `All` or `10` or `/body/input/address`. When available, the policy value takes precedence at run time. If a policy is not found, or is unsuitable, the run time uses the promoted property value.
    c. Click **Add Assertion**.
11. Define one or more gate conditions. Gate conditions must be met before the policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.
    a. Enter a gate condition name in the **Gate condition name** field. The name of a gate condition is always prefixed with the string `medGate_`.
    b. Enter a gate condition value in the **Value** field. The gate condition value is made up of the following parts: *policy condition name*, *operation* and *gate value*.
       - The *policy condition name* you enter must map to a **`Policy condition name`** in the module.
       - The *operation* can be: `=` , `!=` , `>` , `<` , `<=` or `>=` .
       - The *gate value* is the value being compared, for example, `country = France`.
    c. Click **Add Gate Condition**.
12. Optional: If you want to delete an assertion or gate condition, click the delete icon of the appropriate assertion or gate condition. If you hover over an assertion or gate condition, the delete icon, a cross, appears at the end of the row.
13. Click **Save**.

In WSRR, a mediation policy and a policy attachment are created. The policy attachment connects the mediation policy to the service, endpoint, or operation.

For every gate condition you defined, WSRR creates a user property on the policy attachment. The user property represents the condition. In the mediation policy processing model, mediation policies with gate conditions have the highest precedence. Therefore, assertions defined by these mediation policies have the highest precedence. Mediation policies without gate conditions have a lower precedence. Therefore, assertions defined by these mediation policies have a lower precedence. Administrative console values are used if there is no suitable property in a mediation policy.

When the creation process is complete, the widget refreshes, and displays the new policy attachment in the table.

Example: Defining assertions and conditions

The following figure shows two assertions and one gate condition that have been defined, and one gate condition that is in the process of being defined.

Figure 124. Adding assertions and gate conditions, using the Mediation Policy Administration widget

Attaching an existing mediation policy to a target service

Attach existing mediation policies to target services. At the same time, create gate conditions that restrict the use of the mediation policy.

Learn how to reuse existing mediation policies.

5 minutes.

Attaching an existing mediation policy:

You can reuse mediation policies by attaching them to different target services. When you attach an existing mediation policy to a target service, you can create gate conditions that restrict its use.

1. In WebSphere Integration Developer, create conditions on the Policy Resolution mediation primitive. Create a **`Policy condition name`** that matches the gate condition on the mediation policy attachment. Specify where the condition value is found in the message, by providing an XPath expression. For example, you could create a **`Policy condition name`** of Age and an XPath expression of `/body/priceChecker/request/Age`. At run time, the value of condition Age would be set from `/body/priceChecker/request/Age`.
2. Log on to Business Space.
3. Create a business space that contains the Service Browser widget and the Mediation Policy Administration widget.

You might decide to create mediation policies that you reuse when calling certain target services. For example, you might have a mediation policy that turns on logging. However, you might only want to use this mediation policy when calling bank withdrawal operations, and not when calling bank enquiry operations.

You might want to further restrict the logging so that it is only done when a withdrawal is for more than a certain amount. Different WSDL-defined services, endpoints, and operations can use the same mediation policy under different conditions, if you create gate conditions on the policy attachment. At run time, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated policy can be used.

1. From the Service Browser widget, if the correct WSRR definition is not displayed select the correct WSRR definition. If your application server has definitions for more than one instance of WSRR, you can display the services that are defined on each WSRR. The list of services is refreshed.
2. Select the level at which you want to attach a mediation policy. You can attach a mediation policy at the level of the service, endpoint, or operation. The Mediation Policy Administration widget is refreshed. The following information is displayed:
   - The name of the service, endpoint, or operation that you selected.
   - The WSRR definition that you selected.
   - Any policy attachments that exist for the service, endpoint, or operation that you selected.
3. Enter the name of the **New policy attachment**. Policy attachments associate a mediation policy with a target service. In WSRR, the mediation policy and the policy attachment are separate objects.
4. Click **Create**. The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of an existing mediation policy for that group.
5. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
6. Click **Use existing**.
7. Select a mediation policy from the **Select a policy** menu. The mediation policies displayed depend on the group you selected. Because a target service might be called from different modules, the mediation policy associated with the target service might not affect the service request. A mediation policy can only affect a service request if the service, endpoint, or operation that you selected is called by an appropriate module. An appropriate module is one that has properties that the mediation policy can override.
8. Click **Next**. The Mediation Policy Administration widget is refreshed. You can now add gate conditions.

   Note: Using Business Space, you cannot edit mediation policy assertions after you create a mediation policy. However, because gate conditions exist on the mediation policy attachment, you can add gate conditions when you create a new policy attachment.
9. Optional: Define one or more gate conditions. Gate conditions must be met before the policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.
   a. Enter a gate condition name in the **Gate condition name** field. The name of a gate condition is always prefixed with the string `medGate_`.
   b. Enter a gate condition value in the **Value** field. The gate condition value is made up of the following parts: *policy condition name*, *operation* and *gate value*.
      - The *policy condition name* you enter must map to a **`Policy condition name`** in the module.
      - The *operation* can be: `=` , `!=` , `>` , `<` , `<=` or `>=` .

- The *gate value* is the value being compared, for example, `country = France`.

c. Click **Add Gate Condition**.

10. Click **Save**.

In WSRR, a policy attachment is created that associates the selected mediation policy with the selected service, endpoint, or operation.

For every gate condition you defined, WSRR creates a user property on the policy attachment. The user property represents the condition. In the mediation policy processing model, mediation policies with gate conditions have the highest precedence. Therefore, assertions defined by these mediation policies have the highest precedence. Mediation policies without gate conditions have a lower precedence. Therefore, assertions defined by these mediation policies have a lower precedence. Administrative console values are used if there is no suitable property in a mediation policy.

When the creation process is complete, the widget refreshes, and displays the new policy attachment in the table.

Example: Selecting an existing mediation policy

The following figure shows the mediation policies that exist for the group DomainA. You can select one of these mediation policies and attach it to your service, endpoint, or operation.

Figure 125. Selecting an existing mediation policy, to attach to a target service

Tutorial summary

This mediation policy tutorial shows you how to use Business Space widgets to create and reuse mediation policies associated with services. In addition, this tutorial shows you how to use Business Space widgets to create conditions that restrict the use of mediation policies.

After you have completed this tutorial, you can run the application associated with your SCA module. Depending on the message being processed, the target service being called, and on any gate conditions, the mediation flow can be altered by mediation policies.

Tutorial: Administering proxy gateways using Business Space

You can use *proxy gateways* to forward Web service requests to endpoints that you define in *proxy groups*. This tutorial uses Business Space powered by WebSphere to associate existing service endpoints with proxy gateways.

This tutorial shows you how to use a Business Space widget to associate real service endpoints with the proxy groups in your proxy gateway module.

The tutorial is divided into chunks of information called education modules, and each education module has one or more lessons.

Introduction

Because proxy gateways are a type of service gateway, they allow you to access multiple services from one address. They access the services through proxy groups that you define using WebSphere Integration Developer, and administer using Business Space widgets.

Proxy gateways are SCA module containing a Gateway Endpoint Lookup mediation primitive. You can create proxy gateway modules using the Patterns Explorer in WebSphere Integration Developer. When you create a proxy gateway module you define one or more proxy groups. Also, you specify a point in the request message where the name of a *virtual service* can be found. A virtual service is a proxy for one or more real services. You specify whether the virtual service name is found using the URL of the inbound request, which is the default, or an XPath.

After you deploy the proxy gateway module to WebSphere Enterprise Service Bus (WebSphere ESB) or WebSphere Business Process Manager, you can use the Proxy Gateway widget to define the virtual services in the proxy groups. Using the Proxy Gateway widget, you create associations between the virtual services and real service endpoints. The associations are stored in the built-in configuration store.

Learning objectives

- Learn how to configure Business Space for administering proxy gateways.
- Understand how proxy gateways, proxy groups, and virtual services relate to each other.
- Learn how to add virtual services to proxy groups.
- Learn how to add service endpoints to virtual services.
- Learn how to display the details of proxy groups.

This tutorial takes approximately 25 minutes to complete. If you explore other concepts related to this tutorial, it could take longer to finish.

Skill level

Intermediate.

Audience

Integration Developers.

Administrators.

System requirements

- WebSphere Integration Developer, Version 7.0
- WebSphere Enterprise Service Bus, Version 7.0 or WebSphere Process Server, Version 7.0
- Optional: WebSphere Service Registry and Repository, Version 6.2 or above

Prerequisites

1. Read the proxy gateway scenario, which explains the overview. The proxy gateway scenario also explains how the developer and the administrators must work together.
2. Create a proxy gateway module that specifies a **Lookup Method** of **URL**. When the proxy gateway processes service requests part of the input URL is used as the virtual service name.
3. Ensure that the services, and the WSDL documents that define the services, are available.
4. If you have not yet configured WebSphere ESB or WebSphere Business Process Manager for your proxy gateway, do so now. From the administrative console, install your proxy gateway module.

Starting and configuring Business Space

Configure Business Space to administer proxy gateways and proxy groups. This education module shows you how to log on to Business Space and configure the Business Space environment.

Learn how to log on to Business Space and create a business space containing the Proxy Gateway widget.

5 minutes.

Starting Business Space:

Start Business Space and log on.

1. Open a Web browser.
2. Type the Business Space URL. The Business Space URL is: `http://yourhost:yourport/BusinessSpace` where *yourhost* is the IP address of your server, and *yourport* is the port of your server, for the HTTP transport. For example, the URL might be http://localhost:9080/BusinessSpace. The login screen is displayed.
3. If security is enabled, enter a valid user ID and password. The default user ID is admin.
4. Click **Login**.

A Business Space start screen is displayed. The start screen can vary, depending on how you have configured it.

Example: Business Space Login

The following figure shows the Business Space Login screen, if security is enabled.

Figure 126. Business Space Login

Configuring Business Space for proxy gateways:

Create a business space that includes the Proxy Gateway widget.

This lesson is only necessary if you have not worked with proxy gateways before.

Note: When you configure the Business Space tool, you create your own business spaces. For this tutorial, you need a business space that contains the Proxy Gateway widget.

1. Click **Manage Spaces**. The Space Manager opens.
2. Click **Create Space**.
3. Type a name for the new business space.
4. Optional: Type a description for the new business space.
5. Click **Create a new space using a template**, then select **Service Administration**. The Service Administration template creates a business space that contains a Proxy Gateway page, containing the Proxy Gateway widget.
6. Optional: Select a **Space style**.
7. Click **Save**.
8. Click the business space name.
9. Click **Edit Page**.
10. Drag a **Proxy Gateway** widget from the palette to the page.

11. Click the **Save this page** icon.
12. Click **Finish Editing**.

After the browser has been refreshed, your new business space is displayed. The business space has a page for the Proxy Gateway widget.

Example: A business space for proxy gateways

The following figure shows a Proxy Gateway widget containing one proxy gateway and an associated proxy group.

Figure 127. A proxy gateway and a proxy group

Administering proxy gateways and proxy groups

Define virtual services in your proxy groups, and associate the virtual services with real service endpoints.

Learn how to add virtual services and existing service endpoints to proxy groups.

10 minutes.

Adding virtual services to proxy groups:

Use the Proxy Gateway widget to add virtual services to proxy groups.

1. Log on to Business Space.
2. Create a business space that contains the Proxy Gateway widget.
1. From the Proxy Gateway widget, select the **Proxy Group** that you want to work with. Click the pencil icon at the end of the relevant row. The Proxy Gateway widget is refreshed. If the configuration store contains virtual services for the proxy group, the virtual services are displayed. If a virtual service is associated with more than one endpoint, only the first endpoint is displayed in the table.
2. In the **WSDL Location**, enter the location of the WSDL that describes the virtual service. You can provide the WSDL by entering the URL of the service, and appending `?wsdl`. For example, `http://testserver:9080/services/testservice?wsdl` The WSDL might be stored in WSRR or another repository. When the widget is refreshed, the service name in the WSDL is used to populate the **Virtual Service Name** field.
3. Click **Add Service...** The Proxy Gateway widget is refreshed and shows the following information:
   - Port Type: The WSDL portType of the virtual service.

- **Virtual Service Name**: The name of the virtual service that is stored in the configuration store. The default name is entered for you, and is based on the service name in the WSDL you specified.
- Virtual Service URLs:
  - Proxy Gateway: The name of the proxy gateway module.
  - Endpoint: The endpoint of the virtual service.

    Note: Copy the virtual service endpoint. You need the virtual service endpoint to retrieve the WSDL that the client uses.
- **Enable Virtual Service**: A check box that indicates whether you can send messages to the virtual service. By default, virtual services are enabled.
- **Endpoint URLs**: One or more network-addressable endpoints to which a message can be forwarded. If you define a list of endpoints, you can determine the order in which the services are tried. You determine the order by moving the endpoints up and down in the list.
- Advanced Service Properties: If you need to do special processing, for a particular virtual service, you might use the Advanced Service Properties. The Advanced Service Properties are key-value pairs that you want to be accessible in the mediation flow, after the Gateway Endpoint Lookup mediation primitive. For example, you might want to specify the location of the XSL style sheet that relates to this virtual service. At run time, the key-value pairs are stored in the EndpointLookupContext of the service message object (SMO).
  - **Name**: The name of the key.
  - **Value**: The value of the key.

4. Optional: Add an endpoint to the virtual service.
   a. Click **Add Endpoint.**
   b. Enter the URL of the Web service.
5. Optional: Delete one of the endpoints, defined for the virtual service. Click the cross icon at the end of the relevant row.
6. Save the endpoint information. Click **Save**.

   Note: To return to the previous screen, click **Cancel**.

The new virtual service is added to the built-in configuration store, and the Proxy Gateway widget is refreshed.

Example: Adding virtual services to proxy groups

The following figure shows a virtual service that has two Web service endpoint URLs.

Figure 128. Adding virtual services to proxy groups

Displaying and updating proxy gateways and their proxy groups

Display the proxy gateways and proxy groups that have been installed on WebSphere ESB or WebSphere Business Process Manager. Add new endpoints to a virtual service.

Learn how to display proxy gateways and proxy groups, and update virtual services.

5 minutes.

Listing proxy gateways and proxy groups:

Use the Proxy Gateway widget to list the proxy gateways and proxy groups installed on your system.

1. Log on to Business Space.
2. Create a business space that contains the Proxy Gateway widget.
1. Open the business space that you created for proxy groups. Click **Manage Spaces** and click the name of the space that you want to open.
2. If necessary, switch to the proxy group page.

The Proxy Gateway widget lists the following information for each proxy gateway:

- Proxy Group: A group containing service endpoints. If there multiple proxy groups, you can sort the entries. Use the icon after the Proxy Group heading to sort the entries in ascending (^) or descending (v) order.

- Proxy Gateway: A module that can forward Web service requests to an endpoint defined in a proxy group. If there multiple proxy gateways, you can sort the entries. Use the icon after the Proxy Gateway heading to sort the entries in ascending (^) or descending (v) order.

Example: Listing proxy gateways and proxy groups

The following figure shows a proxy gateway that has one proxy group associated with it.

Figure 129. Listing proxy gateways and proxy groups

Displaying proxy group details:

Use the Proxy Gateway widget to display the details of proxy groups. Optionally, add another endpoint to a virtual service.

1. Log on to Business Space.
2. Create a business space suitable for proxy groups, using the Service Administration template.
1. From the Proxy Gateway widget, select the **Proxy Group** whose details you want to display. Click the pencil icon at the end of the relevant row. The Proxy Gateway widget is refreshed.

   If the built-in configuration store contains virtual services for the proxy group, the Proxy Gateway widget displays the following information:

   - Enabled: An icon indicates whether you can send messages to the virtual service. On another panel, you can enable or disable virtual services.
   - Virtual Service: The name of a proxy service. You can associate a virtual service with one or more existing service endpoints.
   - Endpoint: One or more existing service endpoints that are associated with the virtual service.
2. Optional: Add another endpoint to the virtual service.
   a. Click **Add Endpoint.**
   b. Enter the URL of the Web service.
   c. Optional: Move the new endpoint to the top of the list. Click the up arrow.
3. Optional: Delete one of the endpoints, defined for the virtual service. Click the cross icon at the end of the relevant row.
4. Save the endpoint information. Click **Save**.

   Note: To discard the information and return to the previous screen, click **Cancel**.

The new endpoint is added to the virtual service, in the built-in configuration store, and the Proxy Gateway widget is refreshed.

Example: Adding an endpoint to a virtual service

The following figure shows the details of one proxy group, and a Web service endpoint being added.

Figure 130. Adding a Web service endpoint to a virtual service

Retrieving the WSDL with which to call a virtual service

Retrieve the WSDL with which your client can request a virtual service.

Learn how to retrieve the WSDL with which your client can call a virtual service.

5 minutes.

Retrieving the virtual service WSDL:

Use a Web browser to retrieve the WSDL that identifies both a proxy gateway and a virtual service.

1. In Business Space, create a virtual service with an endpoint.
2. Copy the URL of the virtual service endpoint (not the real service endpoint).

Before a client can access a proxy gateway, and have a request routed to a virtual service, it needs the correct WSDL. The WSDL required is constructed by WebSphere ESB or WebSphere Business Process Manager.

1. Open a Web browser.
2. Enter the URL of the virtual service endpoint, appended with the string: `?wsdl`. For example, `http://WebSphereEsb1:9080/BankingProxyGatewayWeb/sca/BankingProxyGatewayExport/BankAccountGoldService?wsdl` where

`BankAccountGoldService` is the name of the virtual service and the rest of the URL is the address of the proxy gateway.

The WSDL is returned.

Tutorial summary

This proxy gateway tutorial shows you how to use a Business Space widget to define virtual services for proxy groups belonging to proxy gateways. In addition, this tutorial shows you how to add endpoints to your virtual services and retrieve the WSDL to call a virtual service.

After you have configured your runtime environment and completed this tutorial, you can run the application associated with your proxy gateway module. Depending on the virtual service name in the message, the proxy gateway routes requests to different service endpoints.

Tutorial: Assessing the health of your solution

This tutorial describes how you can use a business space to monitor the health of your IBM Process Server or WebSphere Enterprise Service Bus solution. The System Health widget provides a single place from which you can quickly assess the status of application servers, nodes, clusters, deployment environments, messaging engines and their queues, databases, system applications, and failed events.

Before you begin

This tutorial uses a widget that contains content from IBM Process Server and WebSphere Enterprise Service Bus. One of those products must be installed. If a product is not installed, Business Space replaces its widgets with an image and a message.

Ensure that you have the appropriate permissions to view and use System Health. This task is typically performed by users with an administrator role.

If you are using System Health in a deployment environment, the deployment manager must be running.

About this task

This tutorial assumes that your daily responsibilities include monitoring the health of your Process Server orWebSphere Enterprise Service Bus solution, making sure that all components are in their expected states (stopped or started). To perform this duty, you use Business Space and complete the following steps.

In this tutorial, you focus only on billing-related applications. All these applications are deployed to clusters in the BillingActivity deployment environment, all contain the word *billing* in their names, and all are deployed to clusters that contain the word *bill* in their names (for example, CustomerBillingCluster1 and VendorBillCluster).

Procedure

1. Log on to Business Space and open the System Health widget.
2. Customize the widget so that it displays only the billing-related components you want to monitor:
   a. Click **Configure** from the widget menu.

b. In the **Display filters** area, customize your display criteria as follows:
   - Select **Deployment environments** and type `BillingActivity` in the box underneath it.
   - Select **Clusters** and type `*bill*` in the box underneath it.
   - Select **Applications** and type `*billing*` in the box underneath it.

c. Clear all the other check boxes in the **Display filters** area.

d. Click **OK**. The System Health widget refreshes and displays the status for the billing-related applications in the specified clusters of the BillingActivity deployment environment, making it simple for you to identify any problems.

3. In the **Topology** tab, verify that the clusters and the deployment environment have the correct status. If any of them are unexpectedly stopped or unavailable, follow your normal troubleshooting procedure.
4. In the **Applications** tab, verify that the billing applications have the correct status. If any of them are unexpectedly stopped or unavailable, follow your normal troubleshooting procedure.

Tutorial: Managing your timetables

This tutorial describes how you can use a business space to view, edit, and modify timetables that define the available time for your business application, and how you can set the appropriate role-based access for the timetables.

Before you begin

This tutorial uses widgets that contain content from IBM Process Server. If IBM Process Server is not installed, Business Space replaces its widgets with an image and a message.

Some of the steps and information might not apply to your situation. Adapt the tutorial to fit your needs.

About this task

In this tutorial, you learn how to use Business Space to work with timetables for your business solution and role-based access to those timetables, using the Business Calendars widget and the Security Roles widget.

You are responsible for updating the available time for a solution that is integrated with a new business partner's human resources application. You must account for the available time for the new business partner by modifying timetables. Also, you must make sure that the appropriate team members have access to the timetables and can modify them in the future.

To perform these tasks, use Business Space with the following widgets: Business Calendars and Security Roles. Complete the following steps:

Procedure

1. Log onto Business Space and open the Business Calendars widget.
2. Your business partner has two additional holidays, December 26 and 27, which you must add to the Holiday timetable. Under **All Timetables**, double-click the Holiday timetable.
3. At the bottom of the Time intervals table, click the **Create Time Interval** button to create a time interval for these two days.

4. Type `business partner holidays` in the subject, and make sure that the **Available time** check box is cleared, because a holiday represents unavailable time. Under **When**, select December 26 from the calendar. Under **Recurrence**, select **Daily**, and under **Every**, select **2**, to designate that the time interval lasts for two days.
5. Click **Save** to save the new time interval for the Holiday timetable.
6. To give all of the human resources employees the ability to view this timetable, complete the following steps:
   a. Open the Security Roles widget.
   b. Click the Reader role for the timetable.
   c. From the **Module Chooser** list, select the module that contains the Holiday timetable.
   d. click the Reader role for the timetable and then perform the following steps.
   e. In the window on the right side of the page, click **Add**.
   f. Type the name (or part of the name) for the group of human resources employees and click **Search**.
   g. From the list that is displayed, select one or more users or groups and click **OK**.
7. To give the human resources manager the ability to edit the timetable, complete the following steps:
   a. Click the Writer role for the timetable.
   b. In the window on the right side of the page, click **Add**.
   c. Type the name (or part of the name) of the manager and click **Search**.
   d. From the list that is displayed, select the user name and click **OK**.
8. Click **Save** to save all of the role assignment changes.

Tutorial: Configure WSRR for use with SLA Endpoint Lookup mediation primitive

This tutorial describes how to configure WSRR for use with the WebSphere ESB SLA Endpoint Lookup mediation primitive.

The tutorial is based on the following scenario:

A web service called Bank Account Application is to be made available to Web service clients (Internet Consumer). The Bank Account Application has been through a development cycle but is only available to external clients when it is in production state. To set up the service in this way, define the service details in WSRR and access the service using WebSphere ESB.

At a high level, the configuration steps are:

- Import the WSDL describing the service into WSRR.
- Create a service level definition (SLD), which defines how to interact with the service. You must specify an entity type and target for the following properties:
  - A service endpoint, a reference to the network addressable endpoint. This endpoint is often a Web service endpoint. See Figure 1.
  - A service interface, a reference to the WSDL port type for the service which defines the valid operations.
  - A service port, a reference to the WSDL port which defines the binding for the service.

- Create a service level agreement (SLA). An SLA defines the level of service that you can expect from the service. For example, there may be policies which restrict the service in some way. An SLA can have an associated **Context Identifier** that can be used to select the SLA.
- Create a service version, using the governance enablement profile (GEP), to reference the SLD and SLA and so provide you with a view of the capabilities of the service. A service version can have an associated **Consumer Identifier**, which represents the Internet Consumer's usage of the Bank Account application under the conditions of the associated SLA.
- Provide an **Endpoint Classification** on the service endpoint to define whether the service is in development or production state.

  Note: There are many other available states, but for the purpose of this tutorial, only development and production state are used.

Figure 131. WebSphere Enterprise Service Bus and WebSphere Service Registry and Repository interaction when creating an endpoint.

The SLA Endpoint Lookup mediation primitive in WebSphere ESB sends a lookup query to WSRR. The query by default contains three query parameters:

- a **Context Identifier**
- a **Consumer Identifier**
- an **Endpoint Classification**.

These parameters correspond to the items configured in the configuration steps. The values for these items are defined by the client.

Introduction

The learning objectives of this tutorial are:

- Understand the relationship between the artifacts in WSRR and the SLA Endpoint Lookup primitive properties.
- Learn how to create a service version in WSRR
- Learn how to create an SLD in WSRR.
- Learn how to create an SLA in WSRR.
- Learn how to classify an endpoint and make that endpoint available.
- Learn how to extend the SLA EndpointLookup query.

Time required

60 minutes

Skill level

Advanced

Audience

Integration Developers.

Administrators.

System requirements

- WebSphere Integration Developer, Version 7.5
- WebSphere Enterprise Service Bus, Version 7.5
- WebSphere Service Registry and Repository, Version 7.5

Prerequisites

1. Read the SLA Endpoint Lookup mediation primitive topic in the Information Center, which explains how the mediation primitive works. It also explains how the developer, the runtime administrator, and the WSRR administrator must work together.
2. See the WSRR documentation on the Governance enablement profile.
3. Log on to the WSRR Web user interface. To start the Web user interface, enter the following URL into the address bar of the Web browser: `http://hostname:port/ServiceRegistry` where *hostname* is the IP address or host name of the WebSphere Application Server that is hosting the WSRR application, and *port* is the server port number.
4. If necessary, switch to the governance enablement profile. By default, WSRR Version 7.5 uses a governance enablement profile; therefore, this tutorial assumes that you are using the governance enablement profile. To check which profile is active, click **Perspective** > **Configuration**; then click **Manage Profiles** > **Configuration Profiles**.
5. If necessary, switch to the WSRR Administrator perspective by selecting to **Perspective** > **Administrator**. This tutorial assumes that you are using the Administrator perspective.

Importing the WSDLs associated with the target services

This module guides you through the process of installing service definitions in WSRR so that they can later be associated with the Government Enablement Profile.

5 - 10 minutes

Importing the WSDLs

1. Go to **View** > **Service Documents** > **WSDL Documents**.
2. Click **Load Documents**. The Load Documents page is displayed.
3. Click **Browse** to find the WSDL file that you want to load.
4. Enter the document description.
5. Click **OK**. A tree showing the WSDL and prerequisite files is displayed.
6. Click **Finish**.

Creating a service version corresponding to the WSDL Description Imported

A service version in the GEP represents a specific version, or release, of a service. In this example, you create a service version which represents the Internet Consumer's usage of the BankAccountService. This module guides you through the creation of a service version using the WSRR Web user interface.

5 - 10 minutes

Creating a service version

1. Switch to the **SOA Governance** perspective.
2. Go to **View** > **Technical Governance** > **Capability Versions** > **Service Versions**.
3. Click **New**.
4. Specify a name for the service version. For example, `BankAccountService`.
5. Optional: If the selection of the SLA is based on the Consumer Identifier, specify a Consumer Identifier. For example, `InternetConsumerBankAccount`.
6. Click **Finish**.

   Note: Later in this tutorial you will return to the service version to associate a Service Level Definition and a Service Level Agreement with it.

Creating a service level definition (SLD)

The SLD provides a specification of the mechanisms used to deliver the messages for a service, and includes non-functional characteristics, such as security and identity. This module describes the process to create the SLD and how to change the SLD state.

10 -15 minutes

Creating an SLD

1. Go to **Select View** > **SOA Governance** > **Service Level Definitions**.
2. Click **New SLD**.
3. Specify a name for the SLD. For example, `BankAccountServiceDefinition`.
4. Click **Add Service Endpoint**.

5. Enter * and click **Search** in the **Search for Existing Entity** name field, to see a list of service endpoints. Leave Entity Type as the default, Service Endpoint.
6. Select the required target. For example, `http://myServer:9080/BankAccountService/BankAccountService` and click **Apply Selected Targets**.
7. Click **Add Service Interface**.
8. Enter * and click **Search** in the **Search for Existing Entity** name field, to see a list of Service Interfaces. Leave Entity Type as the default, Service Interface.
9. Select the required target. For example, `BankAccountServiceInterface` and click **Apply Selected Targets**.
10. Click **Add Service Port**.

11. Enter * and click **Search** in the **Search for Existing Entity** name field, to see a list of Service Ports. Leave Entity Type as the default, Service Port.
12. Select the required target. For example, `BankAccountService_BankAccountServiceHttpPort` and click **Apply Selected Targets**.
13. Click **Finish**. A confirmation message says that the object was successfully created.

Changing the SLD state

For an SLD to become active, the associated SLD must be in a subscribable state. To do make the SLD active, complete the following steps:

1. Click **Propose Scope**. The Governance State is now **Scope Review**.
2. Click **Approve Scope**. The Governance State becomes **Scoped**.
3. Click **Propose Specification**. The Governance State is now **Review**.
4. Click **Approve Specification**. The Governance State becomes **Subscribable** and the SLD is now ready to use.

Creating an SLA

The SLA defines a dependency between a service version and the SLD representing the service provider. This module describes the steps to create the SLA and how to change the SLA state.

10 minutes

Creating an SLA

1. Go to **View** > **SOA Governance** > **Service Level Agreements**.

2. Click **New SLA**.
3. Give a name to the SLA. For example, `BankAccountSLA`.
4. Optional: If the selection of the SLA is based on the Context Identifier, specify a Context Identifier. For example, `BankAccountSLA`.
5. Click **Add Service Level Definition**.

6. In the **Search for Existing Entity** name field, enter * and click **Search**.
7. From the resulting menu select the SLA that you have previously created and click **Apply Selected Targets**.

8. Click **Finish**. The Governance State of the SLA is now Identified.

Changing the SLA state

The SLA must move through a number of states to become active.

1. Click **Request SLA**. The Governance State of the SLA is now Requested.
2. Click **Approve SLA Request**. The Governance State of the SLA becomes Inactive.
3. Click **Activate SLA**. The Governance State of the SLA is now **Active** and it is ready to use.

Associating the SLD and SLA with the service version

The previous modules in this tutorial have defined the SLA and SLD with their relationships, and the definition of the service version. This module describes the process of associating the SLA and SLD with the service version.

Associating the SLD and SLA with the service version

1. Go to **Service Versions**.

2. Select the service version that you created in Module 2 - Creating a service version.
3. Click the **Edit Relationships** link.
4. Select **Add Service Level Definition**.

5. In the **Search for Existing Entity** name field, enter * and click **Search**.
6. From the resulting table select the SLD that you have previously created. For example, `BankAccountServiceDefinition` and click **Apply Selected Targets**. The SLD name is displayed in the **Provides** field.

7. Select **Add Service Level Agreement**.
8. In the **Search for Existing Entity** name field, enter * and click **Search**.
9. From the resulting table select the SLA that you have previously created. For example, `BankAccountSLA` and click **Apply Selected Targets**. The SLA name is displayed in the **Consumes** field
10. Click **Finish**.

Setting the service endpoint to be online and available

The SLA, SLD, and service version have been defined. The endpoint associated with the SLD is in Offline mode by default, and the production classification is not associated. This module shows you how to change the mode to Online, and how to associate the production classification.

Change the endpoint mode to Online then associate the production classification

For an endpoint to be moved into an online state so that it can be used by consumers, it must be classified in terms of its environment.

1. Go to **View** > **SOA Governance** > **SOAP Service Endpoints**.

2. Select the endpoint to activate. For example, `http://myServer:9080/BankAccountServiceWeb/Sca/BankAccountService`.
3. Select **Edit Classifications**.
4. In the Classifications menu, expand **Governance Profile Taxonomy** A further list of options is displayed.
5. Expand **Environment**
6. Select the tick box next to an environment. For example, `Production` and click **Add >>>**.
7. Select the new classification in the **Classification list** and the endpoint for that classification is displayed in the **Details** box. For example, `http://www.ibm.com/xmlns/prod/serviceregistry/6/1/GovernanceProfileTaxonomy#Production`. This is the value that will be used in the SLA Endpoint Lookup Endpoint Classification parameter.

8. Click **OK**.
9. The Classification of the endpoint is currently Offline. Select the **Approve For Use** button.

The Classification becomes Online and the endpoint is now ready for use.

Advanced: Extending the SLA Endpoint Lookup query

15 minutes

Introduction

You can only pass three parameters to WSRR to find an endpoint using the default SLAEndpointLookup query.

Table 19. Default parameters and their assigned values

Parameter name	Value used in Xpath query
ConsumerID	%1
ContextID	%2
Endpoint Classification	%3

In the XPath query (an extract from `SLAEndpointLookup.xml` file) the ConsumerID value substitutes `%1`, the ContextID substitutes `%2` and the Endpoint Classification substitutes `%3`, as shown in this example:

You can specify additional query parameters using the SLA Endpoint Lookup mediation primitive; however, for each additional parameter after the three default parameters represented by %1, %2 and %3, the query must be extended to use the value appropriately. The order that the additional properties are specified in the

table is important. Therefore, the first additional parameter in an updated query must be represented by `%4`, the second additional parameter is represented in the updated query by `%5`, and so on.

This document gives two examples of extending the SLA Endpoint Lookup query: adding an additional endpoint classification, and adding a user property. Ensure that you save any query before making changes to avoid losing your original version. Export the XML that must be edited from WSRR by completing the following steps:

1. Go to the **Configuration** perspective.
2. Select **Active Profile** > **Named Queries**.
3. Select the check box next to the *SLAEndpointLookup* query.
4. Click the **Export** button.
5. Click **Save** and browse to an appropriate location.
6. Click **Save**.

Adding an additional endpoint classification

In this example, a service version named InternetConsumerBankAccount has the endpoint classification Production and the associated SLA has context ID BankAccountSLA. To find this particular endpoint the SLA Endpoint Lookup mediation primitive specifies these three fields to populate the SLA Endpoint Lookup query. For example:

ConsumerID = `InternetConsumerBankAccount`

ContextID = `BankAccountSLA`

Endpoint Classification = `http://www.ibm.com/xmlns/prod/serviceregistry/6/1/GovernanceProfileTaxonomy#Production`

To distinguish this endpoint from other production endpoints the service endpoint also has a Customer Service classification (see the "Edit classifications" step in Module 6 of this tutorial) for instructions on how to assign a new classification to an endpoint). To ensure that the correct production endpoint is selected, add an additional user-defined query parameter to the SLA Endpoint Lookup query to specify the Customer Service classification:

1. Select the **SLAEndpointLookup primitive**.
2. Under **Properties**, select **Details**.
3. Select **Advanced**.
4. Click **Add**.
5. Specify a value to use in the query as parameter 4. In this case `%4`. This value can be an XPath expression referring to a particular value in the service message object (SMO), or it can be a literal string, for example: `http://www.ibm.com/xmlns/prod/serviceregistry/profile/v6r3/BusinessDomain#CustomerService`
6. Optional: Give a description of the parameter. This description is used for documentation only and is not used in the query itself.
7. Click **Finish**.

You must edit and reload the SLAEndpointLookup query into WSRR for this user-defined query parameter to be handled by WSRR. The new parameter relates to endpoint classification so it is the last part of the query to be updated. To ensure that both the original endpoint classification Production and the new classification

CustomerService are defined on the endpoint, change the query from **classifiedByAnyOf** to **classifiedByAllOf**. As the new property is the fourth possible parameter in the query it shows in the XPath query as `%4`. The updated query looks like this:

```
/wsrr:WSRRServer=9Lq2ynf7Ow8Qq4qPE/, /http://www.ibm.com/wsrr/ontology/entity/2ef7c4b5/Governance#classifiedBy#CollapsedFilt?objectUri=* and ppd2_consumes[.] and Bpp2_consumerIdentifier='%2'][ppd2_consumes[.]/[/text()=%4%qPE/], /http://www.ibm.com/wsrr/ontology/entity/2ef7c4b5/Service#ServiceEndpoint#Filt?serviceEndpointUri=* and Bpp2_consumerIdentifier='%2'][ppd2_providesEndpoints[.]/[ppd2_providesEndpoints[.]/[/text()=%4%qPE/]]], /http://www.ibm.com/wsrr/ontology/entity/2ef7c4b5/Service#ServiceEndpoint#Filt?serviceEndpointUri=* and Bpp2_consumerIdentifier='%2'][ppd2_providesEndpoints[.]/[ppd2_providesEndpoints[.]/[/text()=%4%qPE/]]], /http://www.ibm.com/wsrr/ontology/entity/2ef7c4b5/Service#ServiceEndpoint#Filt?serviceEndpointUri=* and Bpp2_consumerIdentifier='%2'][ppd2_providesEndpoints[.]/[ppd2_providesEndpoints[.]/[/text()=%4%qPE/]]]
```

To update the XML the SLAEndpointLookup query, delete it from WSRR and load the new query:

1. Go to the **Configuration** perspective.
2. Select **Active Profile** > **Named Queries**.
3. Select the check box next to the *SLAEndpointLookup* query.
4. Click the **Delete** button.
5. Click **Load Named Query Configuration**.
6. Select the `SLAEndpointLookup.xml` file and ensure that the name is `SLAEndpointLookup`.

   Note: The entry is case-sensitive.
7. Click **OK**.

Adding a user property on the SLD

The SLD provided in the previous example has a custom property **lang**. See Managing custom properties in the WSRR Information Center for instructions on how to create a custom property in WSRR. In this example, during the development phase of the service it only returns text in English and the **lang** property is set to **English**. For production, the service can return text in any of the supported languages and the **lang** property is set to **All**. To specify the value for this property in the lookup query you must add another additional query parameter on the SLA Endpoint Lookup mediation primitive, then change the SLAEndpointLookup query appropriately and reload it into WSRR. To add the new property:

1. Select the *SLAEndpointLookup* primitive.
2. Under **Properties**, select **Details**.
3. Select **Additional Query Parameter**.
4. Click **Add**.
5. Specify a value to use in the query as parameter 5. In this case, `%5`. This value can be an XPath expression that refers to a particular value in the SMO or it can be a literal string. For example, `All`.
6. Optional: Enter a description of the parameter. This description is used for documentation only and is not used in the query itself.
7. Click **Finish**.

The new parameter describes an attribute of the Service Level Definition. Parameter 5 is a further refinement of the search criteria and is associated with the SLD. The full query now becomes:

```
/wsrr:WSRRServer=9Lq2ynf7Ow8Qq4qPE/, /http://www.ibm.com/wsrr/ontology/entity/2ef7c4b5/Governance#classifiedBy#CollapsedFilt?objectUri=* and ppd2_consumes[.] and Bpp2_consumerIdentifier='%2'][ppd2_consumes[.]/[/text()=%4%qPE/]], /http://www.ibm.com/wsrr/ontology/entity/2ef7c4b5/Service#ServiceEndpoint#Filt?serviceEndpointUri=* and Bpp2_consumerIdentifier='%2'][ppd2_providesEndpoints[.]/[ppd2_providesEndpoints[.]/[/text()=%4%qPE/]]], /http://www.ibm.com/wsrr/ontology/entity/2ef7c4b5/Service#ServiceEndpoint#Filt?serviceEndpointUri=* and Bpp2_consumerIdentifier='%2'][ppd2_providesEndpoints[.]/[ppd2_providesEndpoints[.]/[/text()=%4%qPE/]]]
```

To update the XML the SLAEndpointLookup query, delete it from WSRR and load the new query:

1. Go to the **Configuration** perspective.
2. Select **Active Profile** > **Named Queries**.
3. Select the check box next to the *SLAEndpointLookup* query.
4. Click the **Delete** button.

5. Click **Load Named Query Configuration**.
6. Select the `SLAEndpointLookup.xml` file and ensure that the name is `SLAEndpointLookup`.

   Note: The entry is case-sensitive.
7. Click **OK**.

Tutorial summary

Figure 132 shows the suggested end-to-end scenario for SLA endpoint resolution. The Service Consumer populates header information within the message that describes the context and consumer IDs for the particular request (typically using the GEPHeader structure supplied with WebSphere ESB). When the message is received by WebSphere ESB within the Mediation Flow Component the context and consumer IDs are extracted from the header. The SLAEndpoint Named Query is then submitted to WSRR using the endpoint classification specified on the SLA Endpoint Lookup mediation primitive. This query information is used by WSRR to determine the endpoint information of the Service Providers that meet the selection criteria and is returned to WebSphere ESB for routing the request. Additional parameters can be submitted to WSRR to further restrict the returned endpoints; however, you must populate these parameters based on either content of the service message object (SMO) or static content defined in the mediation primitive. Additionally, the named query within WSRR must be updated to use this information.

Figure 132. The suggested end-to-end scenario for SLA endpoint resolution

This completes the tutorial for the creation of the governance enabled profile artifacts within WSRR for usage with the SLA Endpoint Lookup mediation primitive.

Chapter 5. Administering the IT infrastructure

Administering the IT infrastructure of WebSphere ESB involves preparing, monitoring, and modifying the runtime environment into which applications, their modules, and resources are deployed.

The topics in this section describe how to administer the infrastructure for the WebSphere ESB runtime environment, including the modules, environment, and deployed resources.

Related tasks:

Troubleshooting administration
Use these topics to help you identify and resolve problems that can occur during typical administration tasks or within the service applications you are administering.

Related information:

PDF documentation
WebSphere Enterprise Service Bus documentation (in PDF format)

Information roadmaps
Business Process Management information roadmaps on IBM developerWorks organize information about WebSphere Process Server, WebSphere ESB, and the other products in the portfolio.

IBM Education Assistant
Multimedia educational modules about WebSphere ESB, provided by IBM Education Assistant.

Technotes
WebSphere ESB Support > technote search for 6.2 administering category documents. Use the document type, product category, and search terms fields to find the information you need.

Overview
Overview tab, on product library Web page. Use this page to access announcements, data sheets, and other general library documents related to WebSphere ESB.

How do I...? Administer the IT infrastructure for WebSphere ESB

Follow these shortcuts to get started quickly with some common IT administration tasks.

How do I use the administrative interfaces?

- The administrative console

How do I manage the administrative architecture?

- Create a new server
- Start and stop deployment managers, node agents, deployment environments, clusters, and servers
- Modify the deployment topology
- Edit the data source configuration for a deployment environment

© Copyright IBM Corp. 2007, 2011

How do I administer applications?

- Start or stop a service module
- Administer service modules in the administrative console
- Administer service modules in Business Space
- Use Business Space to administer services and their mediation policies
- Manage access to WSRR
- Administer bindings
  - Administer SCA bindings
  - Administer Web service bindings
  - Administer HTTP bindings
  - Administer EJB bindings
  - Administer EIS bindings
  - Administer JMS bindings
  - Administer Generic JMS bindings
  - Administer WebSphere MQ JMS bindings
  - Administer WebSphere MQ bindings

How do I manage relationship runtime data for my application?

- View relationships managed by the relationship service
- View relationship details
- View role details
- Query relationship instance data
- Edit relationship instance details
- Edit role instance data

How do I work with sequenced or failed events?

- Find failed events
- Resubmit failed events

How do I administer adapters?

- Administer WebSphere Business Integration Adapters

Overview of administering WebSphere ESB

Administering WebSphere ESB involves preparing, monitoring, and modifying the environment into which applications and resources are deployed, and managing those applications and resources. Use the following topics to learn more about the interfaces and configuration files used for administration tasks.

The administrative console

The administrative console is a browser-based interface used to administer applications, services, and other resources at a cell, node, server, or cluster scope. You can use the console with stand-alone servers and with deployment managers that manage all servers in a cell in a networked environment.

Note: The administrative console is part of the Integrated Solutions Console framework in general, and the WebSphere Application Server administrative console in particular. As a result, many administrative tasks (for example, setting security, viewing logs, and installing applications) are the same for all products

that use the console, including Process Server and WebSphere Enterprise Service Bus. Those tasks are documented in the WebSphere Application Server Information Center.

If you have installed a stand-alone profile, you have a single node in its own administrative domain, known as a cell. Use the administrative console to manage applications, buses, servers, and resources within that administrative domain.

Similarly, if you have installed and configured a network deployment cell, you have a deployment manager node and one or more managed nodes in the same cell. Use the administrative console to manage applications, set up managed nodes in the cell, and monitor and control those nodes and their resources.

In the administrative console, task filters provide a simplified user experience and, through the progressive disclosure of functions, access to the full underlying WebSphere Application Server administrative capabilities.

Administrative console areas

Use the administrative console to create and manage objects such as resources, applications, and servers. Additionally, use the administrative console to view product messages.

This topic describes the main areas that are displayed on the administrative console.

To view the administrative console, ensure that the server for the administrative console is running. If you have configured a stand-alone server, the console runs on that server. If you have configured a network deployment cell, the console runs on the deployment manager server.

Point a web browser at the address for the administrative console, enter your user ID and, if security is enabled, a password on the Login page.

You can resize the width of the navigation tree and workspace simultaneously by dragging the border between them to the left or the right. The change in width does not persist between administrative console user sessions.

The console has the following main areas.

Taskbar

The taskbar offers options for logging out of the console, accessing product information, and accessing support.

Navigation tree

The navigation tree on the left side of the console offers links to console pages that you use to create and manage components in a cell.

Click a plus sign (**+**) next to a tree folder or item to expand the tree for the folder or item.

Click a minus sign (**-**) to collapse the tree for the folder or item.

Click an item in the tree view to display its console page. This also toggles the item between an expanded and collapsed tree.

Workspace

The workspace on the right side of the console contains pages that you use to create and manage configuration objects such as servers and resources.

Click links in the navigation tree to view the different types of configured objects.

Within the workspace, click configured objects to view their configurations, runtime status, and options. Click buttons to perform actions on selected objects.

Click Welcome in the navigation tree to display the workspace home page, which contains links to product-specific welcome pages for each product that you have installed. Use these pages to see detailed information about each product.

The welcome page also provides a task filtering selector to help refine the administrative console pages. Each filter provides a subset of administrative console functionality pertinent to a particular set of tasks (for example, process server administration or enterprise service bus administration).

Business integration areas of the administrative console

The business integration resources used by IBM Process Server and WebSphere Enterprise Service Bus are grouped into several areas of the administrative console.

Use the navigation tree to locate business integration resources, as follows.

- **Servers > Deployment Environments**: Provides access to manage deployment environments, as well as a wizard to help you create a new deployment environment.

  This option is available only if you have installed WebSphere Application Server Network Deployment.
- **Servers** > **Server Types** > **WebSphere application servers** > *server_name*: Provides access to the following:
  - Business Integration configuration (tabbed page of deployment target functions)
  - Business Space configuration
  - REST Services configuration
  - Service Component Architecture configuration
  - Common Event Infrastructure server and destination configuration
  - Selectors
  - WebSphere Business Integration Adapter Service
  - Application Scheduler
- **Servers** > **Clusters** > **WebSphere application server clusters** > *cluster_name*: Provides access to the following:
  - Business Integration configuration (tabbed page of deployment target functions)
  - Business Space configuration
  - REST Services configuration
  - Service Component Architecture configuration
  - Common Event Infrastructure server and destination configuration
- **Applications** > **SCA Modules**: Provides access to the following:
  - SCA modules and their associated service applications
  - SCA module imports, including interfaces ad bindings

  - SCA module exports, including interfaces and bindings
  - SCA module properties
- **Resources** : Provides access to the following:
  - WebSphere Business Integration Adapters
  - Remote Artifacts
- **Integration Applications**: Provides access to the following:
  - Failed event manager
  - Relationship manager
  - Common Base Event Browser
- **Service integration**: Provides access to the following:
  - WebSphere Service Registry and Repository (WSRR) definitions
  - Service Integration Bus Browser

Administrative console guided activities

Guided activities lead you through common administrative tasks that require you to visit multiple administrative console pages.

Guided activities display each administrative console page that you need for a specific task, surrounded by the following information to help you:

- An introduction to the task and its essential concepts
- A description of when and why to do this task
- A list of other tasks to do before and after the current task
- The main steps to complete during the task
- Hints and tips to help you avoid or recover from problems
- Links to field descriptions and extended task information in the online documentation

Figure 133 on page 360 shows an example of the administrative console displaying a guided activity.

Figure 133. A guided activity

Administrative console pages

Administrative console pages are formatted in one of three ways: Collection, detail, and wizard pages. Understanding the layout and behavior of each type of page can help you use them more effectively.

- "Collection pages"
- "Detail pages" on page 361
- "Wizard pages" on page 361

Collection pages

A collection page manages a collection of existing administrative objects (for example, relationships, failed events, or resource adapters). It contains one or more of the following elements:

Scope and preferences
The scope and preferences help determine which administrative objects are displayed in the table, and how they should appear.

Table of existing objects
The table displays existing administrative objects of the type specified by the collection page. The table columns summarize the values of the key settings for these objects. If no objects exist yet, the table is empty. Use the available buttons to create a new object.

Buttons for performing actions
The typical buttons are described in "Administrative console buttons" on page 361. In most cases, you need to select one or more objects in the collection table, then click a button. The action is applied to all selected objects.

Sorting toggle buttons
After each column heading in the table are icons to sort the entries in

ascending (^) or descending (v) order. By default, items such as object names are sorted in descending order (alphabetically).

Detail pages

A detail page is used to view details about an object and to configure specific objects (such as an application server or a listener port extension). It typically contains one or more of the following elements:

Configuration tabbed page
This tabbed page is used to modify the configuration of an administrative object. Each configuration page has a set of general properties specific to the object. Additional properties can be displayed on the page, depending on the type of administrative object you are configuring.

Changes to this tabbed page can require a server restart before they take effect.

Runtime tabbed page
This tabbed page displays the configuration that is currently in use for the administrative object. It can be read-only. Note that some detail pages do not have runtime tabs.

Changes to this tabbed page take effect immediately.

Local topology tabbed page
This tabbed page displays the topology that is currently in use for the administrative object. View the topology by expanding and collapsing the different levels of the topology. Note that some detail pages do not have local topology tabs.

Buttons for performing actions
Buttons to perform specific actions display only on configuration tabbed pages and runtime tabbed pages. The typical buttons are described in "Administrative console buttons."

Wizard pages

Wizard pages help you complete a configuration process consisting of several steps. Be aware that wizards can show or hide certain steps, depending on the characteristics of the specific object that you are configuring. See "Administrative console guided activities" on page 359.

Administrative console buttons

The administrative console interface contains a number of buttons, depending on which page you are currently viewing. This topic describes the available console buttons.

The graphical buttons in Table 20 are located at the top of a table that displays server-related resources:

Table 20. Graphical buttons at the top of a console collection page

Button	Resulting action
Check all	Selects each resource (for example, a failed event or a relationship instance) that is listed in the table, in preparation for performing an action against those resources.
Uncheck all	Clears all selected resources so that no action is performed against them.

Table 20. Graphical buttons at the top of a console collection page (continued)

Button	Resulting action
Show the filter view	Opens a dialog box to set a filter. Filters are used to specify a subset of resources to view in the table.
Hide the filter view	Hides the dialog box used to set a filter.
Clear filter value	Clears all changes made to the filter and restores the most recently saved values.

The graphical buttons in Table 21 appear at the end of an administrative console page. Not all buttons appear on all pages.

Table 21. Buttons at the bottom of a console page

Button	Resulting action
Add	Adds the selected or typed item to a list, or produces a dialog box for adding an item to a list.
Apply	Saves your changes to a page without exiting the page.
Back	Displays the previous page or item in a sequence. The administrative console does not support using the Back and Forward options in the web browser, which can cause intermittent problems. Use the **Back** or **Cancel** buttons in the console instead.
Cancel	Exits the current page or dialog box, discarding all unsaved changes. The administrative console does not support using the Back and Forward options in the web browser, which can cause intermittent problems. Use the **Back** or **Cancel** buttons in the console instead.
Clear	Clears your changes and restores the most recently saved values.
Clear selections	Clears any selected cells in the tables on this tabbed page.
Close	Exits the dialog.
Delete	Removes the selected instance.
OK	Saves your changes and exits the page.
Reset	Clears your changes on the tab or page and restores the most recently saved values.
Save	Saves the changes in your local configuration to the master configuration.

For a complete list of buttons used in the administrative console to administer all products and resources, refer to Administrative console buttons in the WebSphere Application Server Information Center.

Command-line tools, scripts, and programming interfaces

WebSphere ESB provides command-line tools, scripting interfaces, and programming interfaces to administer the runtime environment.

Command-line tools

Command-line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks. Using these tools, you can start and stop application servers, check server status, add or remove nodes, and other tasks.

The WebSphere ESB command-line tools include the **`serviceDeploy`** command, which processes .jar, .ear, .war and .rar files exported from an Integration Designer

environment and prepares them for installation to the production server.

Scripting (wsadmin)

The WebSphere administrative (wsadmin) scripting program is a non-graphical command interpreter environment that enables you to run administrative options in a scripting language and to submit scripting language programs for execution. It supports the same tasks as the administrative console. The wsadmin tool is intended for production environments and unattended operations.

Administrative programming interfaces

Administrative programming interfaces are a set of Java classes and methods under the Java Management Extensions (JMX) specification that provide support for administering Service Component Architecture (SCA) and business objects. Each programming interface includes a description of its purpose, an example that demonstrates how to use the interface or class, and references to the individual method descriptions.

Configuration information

Configuration data for WebSphere ESB is stored in XML files, which are kept in directories in the configuration repository tree (the master repository).

The directory in which a configuration file exists determines its scope, or how broadly or narrowly that data applies.

- Files in an individual server directory apply to only that server.
- Files in an application directory apply to only that application.
- Files in a cluster-level directory apply to only that cluster.
- Files in a node-level directory apply to every server on that node.
- Files in a cell directory apply to every server on every node within the entire cell.

Table 22. WebSphere ESB configuration files

Configuration file	Description
`server-wbi.xml`	Identifies a server and its components, including WebSphere Business Integration Adapter Service configuration.
`resources-wbi.xml`	Defines operating environment resources for WebSphere ESB and is present at the cell, node, and server scopes. This includes WebSphere Business Integration Adapters.
`cell-wbi.xml`	Identifies a cell.
`server-core.xml`	Identifies configuration information for core WebSphere ESB configurations.

WebSphere ESB configuration files can be edited through the administrative console, wsadmin, and scripting. No manual editing is required.

See Configuration file descriptions for more information.

Configuring the business object parsing mode

You can set the business object parsing mode (the XML parsing mode for business objects) only by using Integration Designer. You can only view the parsing mode for an application by using either the administrative console or a command with WebSphere ESB.

A module can be configured to run in eager or lazy XML parsing mode for business objects. In eager mode, XML data is processed as soon as it is available. In lazy mode, XML data processing time and storage is minimized by applying strategies such as delayed parsing until access, copy on write and serializing the original, unprocessed bytes, where possible. In either XML parsing mode, non-XML data is always eagerly parsed to create the business object.

Use Integration Designer to configure an application to use either lazy or eager parsing. For more details about setting the business object parsing mode, see Configuring the business object parsing mode of modules and libraries. The parsing mode setting you choose is saved in the root directory of the module or library project as `sca_metata.xml`.

Important: If you select lazy parsing mode, consider adding explicit data validation that is executed immediately upon data arrival, using the SCA data validation qualifier or business object XML document validation.

Viewing the Business Object Framework Parsing Mode using the administrative console

Use the administrative console to view the Business Object Framework parsing mode for SCA modules.

Procedure

1. Navigate to **Applications** > **SCA Modules** > *SCA_module_name*.
2. In the **Business Object parsing mode** field, you can see which parsing mode is used for the module. The mode is either `Lazy` or `Eager`.

Viewing the Business Object Framework Parsing Mode using a command

You can use a command to view the Business Object Framework parsing mode for modules.

About this task

Perform the following steps to view whether a module parsing mode has been set to Eager or Lazy:

Procedure

1. This module uses eager business object parsing mode:

```
wsadmin>$AdminTask showSCAModule {-moduleName k6}
name:k6
description:Module using Eager
businessObjectRuntimeVersion: 6
businessObjectParsingMode: Eager
```

2. This module uses lazy business object parsing mode:

```
wsadmin>$AdminTask showSCAModule {-moduleName k7}
name:k7
description:Module using Lazy
businessObjectParsingMode: Lazy
```

Related reference:

showSCAModule command

Getting started with the administrative console

Use the tasks in this topic to get started using the administrative console to manage and administer WebSphere ESB resources.

The following tasks help you start the server and the administrative console, set the console scope and preferences, and save your work to the master repository.

- **Start the server.**

  Before you can use the administrative console, you must start the stand-alone server or deployment manager. For instructions on starting a stand-alone server, see Starting and stopping stand-alone servers. For instructions on starting the deployment manager, see Starting and stopping the deployment manager.
- **Start the administrative console.**

  See “Starting and stopping the administrative console” for details.
- **Specify console preferences.**

  Preferences control how data is displayed in the administrative console, as well as how the workspace behaves. See “Setting administrative console preferences” on page 367.
- **Set the console scope.**

  The scope specifies the level at which a resource is visible on the administrative console. A resource can be visible in a console collection table at the cell, node, cluster, or server scope. See Administrative console scope settings for details.
- **Create filters to view information.**

  Filters specify which data is shown in a column on a collection page. See “Setting administrative console filters” on page 368.
- **Optional: Set the session timeout for the console.**

  By default, a console session times out after 30 minutes of inactivity. You can change this value by editing and running a script, as described in Changing the console session expiration.
- **Save your work to the master repository.**

  Until you save your changes to the master repository, the console uses a local workspace to track the changes. To save your changes, click **System Administration > Save Changes to Master Repository** to display the Save page, and then click **Save**.

Starting and stopping the administrative console

To access the administrative console, you must start it and then log in. After you finish working in the console, save your work and log out.

Before you begin

Ensure you have started the application server required by the administrative console.

About this task

To start the console, log in, and then log out, use the following procedure.

Procedure

1. Start the administrative console:
   a. Enable cookies in the Web browser that you plan to use to access the administrative console.
   b. Optional: Enable JavaScript. JavaScript enablement is recommended so that all the features of the administrative console are available to you.
   c. In your cookie-enabled Web browser, type the following:

      `http://`*your_fully_qualified_server_name*`:`*portNumber*`/ibm/console`

      where *your_fully_qualified_server_name* specifies the fully qualified host name for the system that contains the administrative server and *portNumber* is the administrative console port number. When the administrative console is on the local system, *your_fully_qualified_server_name* can be `localhost` unless security is enabled.

      On Windows platforms, use the actual host name if `localhost` is not recognized.

      If security is enabled, your request is redirected to https://*your_fully_qualified_server_name*:*secure_portNumber*/ibm/console, where *your_fully_qualified_server_name* is the fully qualified host name for the system that contains the administrative server and *secure_portNumber* is the administrative console secure port number.

      Note: The default port number for an unsecure administrative console is port 9060, and for a secure administrative console the default port number is 9043. Each new administrative console that you deploy during profile creation is assigned a new unsecure port number and, if you enable security during profile creation, a new secure port number.
   d. Check the `System.Out.log` file of the server that runs the console application to verify that the console application has started successfully. A successful start produces the message `WSVR0221I: Application started: isclite`.

      If you are unable to start the console because the console port conflicts with an application that is already running on the system, change the port number in the following files:
      - *profile_root*`/config/cells/`*cell_name*`/nodes/`*node_name*`/serverindex.xml`
      - *profile_root*`/config/cells/`*cell_name*`/virtualhosts.xml`

      Change all occurrences of the port selected during profile creation (by default, 9060) to the port for the console. Alternatively, shut down the other application that uses the conflicting port before starting the administrative console.

   The administrative console loads in the browser, displaying a login page.
2. Log into the console:
   a. In the **User ID** field, enter your user name or user ID. The user ID lasts only for the duration of the session for which it is used to log in.

      Note: If you enter an ID that is already in use (and in session) you are prompted to do one of the following:
      - Log out the other user with the same user ID. You can recover changes made during the other user's session.

- Return to the login page and enter a different user ID.

  Any changes made to server configurations are saved to the user ID. Server configurations are also saved to the user ID if a session times out.

b. If security is enabled for the console, you must also enter a password in the **Password** field.

c. Click **OK**.

   The administrative console now displays the Welcome page.

3. Log off the console:
   - To save the work you have done during this session, click **System administration > Save changes to master repository > Save**, and then click **Logout** to exit the console.
   - To exit the console without saving your changes to the repository, click **Logout**.

   If you close the browser before saving your work, you can recover any unsaved changes the next time that you log in with the same user ID.

Related tasks:

"Starting managed servers" on page 382
Start the application server process in order to run applications on the managed server.

Setting administrative console preferences

The display of data on a collection page (a page that lists collections of data or resources in a table) can be customized through administrative console preferences. Preferences are set on a user level, and typically must be set separately for each area of the administrative console.

About this task

You can set the following display preferences for collection pages:

- **Maximum rows**: Specifies the maximum number of rows that are displayed when the collection is large. If there are more rows than the specified maximum, they are displayed on subsequent pages. The default value is 20.
- **Retain filter criteria**: Specifies whether the last search criteria entered in the filter function is retained. If this is enabled, the console collection pages initially use the retained filter criteria to display the data in the table following the preferences. See "Setting administrative console filters" on page 368 for more information.
- **Max result set size**: Specifies the maximum number of resources that a search can return. The default value is 500.
- **Max column width**: Specifies the maximum number of characters viewable in a collection column. The default value is 18.

Perform the following steps to set display preferences for a collection page.

Procedure

1. From any collection page, click **Preferences**.

   The page expands to display the preference fields.
2. Modify the values for the **Maximum rows**, **Retain filter criteria**, **Max result set size**, and **Maximum column width** fields as desired.
3. Click **Apply**.

The collection table is refreshed to display according to the values you specified.

What to do next

You can also set global administrative console preferences, such as whether the workspace is automatically refreshed and which scope to use by default. To access the Preferences page in the administrative console, click **System administration** > **Console settings** > **Preferences**. For more information about setting these preferences, see the WebSphere Application Server Information Center.

Setting administrative console filters

Each table on a collection page in the administrative console displays a list of WebSphere ESB data or resources. You can use a filter to specify exactly which resources or data to display in a particular column of the table. Filters can be set on a single column only.

Procedure

1. Click **Filter the view**.

   The filter dialog box opens above the table.
2. Use the **Filter** drop-down menu to select the column you want to include in the filter.
3. In the **Search terms** field, specify the filter criteria.

   The criteria is a string that must be found in the name of a table entry in order for it to be displayed. The string can contain the percent sign (%), asterisk (*), or question mark (?) symbols as wildcard characters. For example, on the Resource Adapters page, you can enter `*JMS*` as the filter criteria for the Name column to find any resource adapter whose name contains the string `JMS`.

   Prefix each of the following characters that appear as part of the string with a backslash (\) so that the regular expression engine performing the search correctly matches the search criteria: `( ) ^ * % { } \ + & .`

   For example, if you want to search for all Java DataBase (JDBC) providers containing `(XA)` in the provider name, specify the following string in the Search term(s) field:

   ```
   *\(XA\)*
   ```
4. Click **Go**.

   The table refreshes, and only those items in the selected column that meet the filter criteria are displayed.

Using My tasks

Customize console navigation by creating and editing a task view.

About this task

Use **My tasks** to create and edit a list of tasks to view in the console navigation. A task includes a page that contains one or more Web applications, or console modules, that are used to complete that task. When you first access the console, all tasks to which you have access are displayed in the navigation. **My tasks** is especially useful to customize the navigation to show only the tasks you use most often. After you customize your tasks, **My tasks** is initially displayed each time you log in to the console.

Procedure

1. Click the **Welcome** link in the navigation tree.
2. Select **My tasks** from the **View** selection list in the navigation. If you have never used **My tasks** before, you must click **Add tasks** to open it.
3. Select the tasks you want to add to **My tasks** list.
4. To save your changes, click **Apply**.
5. To cancel your changes, click **Reset**.

Results

After you click **Apply**, your customized task list is displayed in the navigation. You do not have to shut down and restart the administrative console.

Accessing product information and help from the administrative console

The administrative console provides access to product documentation as well as online help for each page and field. You can view the help in the console help browser or in the WebSphere ESB Information Center.

About this task

Perform the following steps to access product information and administrative console help topics.

Procedure

1. Access the product information by doing the following tasks.
   a. Click **Welcome** on the administrative console navigation tree. In the workspace to the right of the navigation tree, the console displays information about the installed products.
   b. Click the appropriate links to access the product Information Center and the related technical information on developerWorks.
2. Access the product help in one of the following ways.

Option	Description
Access field-level help in the administrative console	• Place the cursor over a field to view hover help about that field. • Place the cursor over a field and wait for the question mark (**?**) icon to appear. When the icon appears, click the field name to display brief help about it in the Help portal (the right-most panel in the workspace). **Note:** If you want to view extended information about the field, or about the entire page and its associated tasks, click the **More information about this page** link at the bottom of the help portal.

Option	Description
Access the stand-alone help browser	Click **Help** from the console task bar to view online help in a new Web browser. From here you have the following options: • Browse for the topic you want to view in the Index tab. Click the link for that topic to open it in the right panel of the browser. • Search for a topic by specifying one or more key words in the Search tab. All matching topics are displayed in the navigation tree; click a topic link to view it.
View the online help in the WebSphere ESB Information Center	• Use a browser to navigate directly to the WebSphere ESB Information Center. Online help topics are in the **Reference** section. • Click the **Check for updates to this topic** link in any help file viewed with the help browser.
View command assistance	If command assistance is available, click **View administrative scripting command for last action** in the right corner of Help portal.

Related tasks:

"Accessing command assistance from the administrative console"
Use command assistance to see wsadmin scripting commands that correspond to actions in the administrative console. Seeing the commands can help you develop the command-line tools needed to administer the server from the wsadmin utility.

Accessing command assistance from the administrative console

Use command assistance to see wsadmin scripting commands that correspond to actions in the administrative console. Seeing the commands can help you develop the command-line tools needed to administer the server from the wsadmin utility.

Before you begin

Before using command assistance, do the following:

- Start WebSphere ESB and the administrative console.
- Determine whether you want to save command assistance data to a log file. When logging is enabled, a timestamp and the breadcrumb trail of the page that produced the command assistance data are provided with the wsadmin data in the `commandAssistanceJythonCommands_username.log` file in the logs directory for the process running the console.

  Click **System administration** > **Console preferences** > **Log command assistance commands** to save command assistance data to the log file.
- Determine whether you want to allow command assistance to emit Java Management Extensions (JMX) notifications. Enabling the notifications allows integration with product tools that can help you write automation scripts (for example, the WebSphere Application Server Toolkit Jython editor). The notification type is websphere.command.assistance.jython.*user_name*, where *user_name* specifies the current administrative console user.

Note: This option is recommended for non-production environments only.

Click **System administration** > **Console preferences** > **Enable command assistance notifications** to enable JMX notifications.

About this task

Using command assistance, you can view wsadmin scripting commands in the Jython language for the last action run in many pages in the administrative console.

If a command assistance link is listed in the help portlet, wsadmin commands exist for the last console action you completed, and command assistance is available for that action.

Examples of actions include a click on a button or a click on a link in the navigation bar, collection page, or detail page. Editing a form is not a user action and is not captured by command assistance.

The wsadmin scripting commands display in the Jython language in a secondary window. If you perform an administrative action after you launch the Administrative Scripting Commands window, the window automatically refreshes the command list to reflect the most recent console action.

When command assistance is unavailable in the help portlet: Some console actions do not have wsadmin commands directly associated with them. When the help portlet on the right side of the administrative console page does not have a command assistance link in it, no command assistance data is available for the last console action.

To use command assistance in the console, perform the following steps.

Procedure

1. Optional: Set console preferences to capture command assistance data in a log file, as follows:
   a. Click **System Administration** > **Console Preferences** to open the Preferences page.
   b. Select **Log command assistance commands**.
2. Optional: Set console preferences to allow command assistance to emit Java Management Extensions (JMX) notifications, as follows:
   a. Click **System Administration** > **Console Preferences** to open the Preferences page.
   b. Select **Enable command assistance notifications** to emit websphere.command.assistance.jython.*user_name* notifications.
3. Navigate to the console page you want to use with command assistance.
4. Click **View administrative scripting command for last action** from the Help portlet on the right side of the page. The Administrative Scripting Commands window opens and displays the Jython for the related wsadmin scripting command.
5. Optional: View the description of a specific wsadmin command by placing your cursor over the command to display hover help.

Results

You have viewed wsadmin scripting commands from the administrative console, optionally logged the commands to a file, and optionally allowed command assistance to emit JMX notifications.

What to do next

You can use the information provided by command assistance when creating wsadmin scripts to automate administrative tasks.

Related information:

Administrative console actions with command assistance

Administrative console actions with command assistance (WebSphere Application Server)

Using scripting (wsadmin)

Managing the administrative architecture

After you install and configure a deployment environment, use the administrative tools to monitor and control the resources in the deployment environment, including deployment managers, node agents, and clusters.

Starting deployment managers

The deployment manager is a server process. You must start the deployment manager before you can use its administrative console to manage the cell.

About this task

Perform the following steps to start and stop a deployment manager.

Procedure

1. Start the deployment manager with one of the following actions:
   - Windows From the **Start** menu, select **IBM WebSphere** > **Enterprise Service Bus** > **Profiles** > *profile_name* > **Start the deployment manager**.
   - In the First steps console, click **Start the deployment manager**.
   - Use the **startManager** command.
2. Verify that the deployment manager started successfully by checking the `install_root/profiles/profile_name/logs/server_name/startServer.log` log file for the message `Server server_name open for e-business; process id is nnnn`.

What to do next

You can now start the administrative console and manage the cell.

Stopping a deployment manager

Stop the deployment manager server process when performing certain maintenance activities such as migrating to a new version of the product or uninstalling the product. You can stop the deployment manager at any time without affecting the operation of the servers in its domain.

Before you begin

The deployment manager must be running.

About this task

To stop a deployment manager, perform the following steps.

Procedure

1. Stop the deployment manager with one of the following actions:
   - Windows From the **Start** menu, click **IBM WebSphere** > **Enterprise Service Bus** > **Profiles** > *profile_name* > **Stop the deployment manager**.
   - In the First steps console, click **Stop the deployment manager**.
   - From the administrative console, click **System administration** > **Deployment manager** > **Stop** > **OK**. The administrative console closes before the server stops running.
   - Use the **stopManager** command.
2. If you used the stopManager command, verify that the deployment manager stopped successfully by checking the *install_root*/profiles/*profile_name*/logs/*server_name*/stopServer.log log file for the message Server *server_name* stop completed.

Starting node agents

The node agent of a managed node is a server process that must be started before you can start servers on the node. The node agent must be started for the deployment manager to communicate with it.

Before you begin

Before you can start and stop a node, you must federate the node into a cell.

About this task

You must start the node agent from the command line of the host on which the node is configured, in the *install_root*/bin directory.

To start a node agent, perform the following steps.

Procedure

1. Verify that the node agent is not currently running:
   a. Start the administrative console on the deployment manager.
   b. Click **System administration** > **Node agents** and verify that the node agent is stopped.
2. Use the the **startNode** command to start the node agent.
3. Verify that the server started successfully by checking the *install_root*/profiles/*profile_name*/logs/*server_name*/startServer.log log file for the message Server nodeagent open for e-business; process id is nnnn.

Example

- To start the node agent in the default profile, type startNode
- To list the options, type startNode -help

- To start the node agent in the Custom03 profile, type `startNode -profileName Custom03`
- To start the node agent in the Custom03 profile and write trace information to the log file called *install_root*/profiles/Custom03/logs/startServer.log, type `startNode -logfile -profileName Custom03`

What to do next

You can now manage this node from the deployment manager, including starting the servers on the node.

Stopping a node agent

Use the administrative tools when you need to stop a node agent (for example, to change the system clock). Node agents are administrative agents that represent a node to your system and manage the servers on that node.

Before you begin

Stop all servers that are managed by the node agent before you stop the node agent.

About this task

To stop a node agent, perform the following steps.

Procedure

1. From the administrative console of the deployment manager, click **System administration** > **Node agents**.
2. Select the node agent from the list on the Node Agent collection page.
3. Click **Stop**.

What to do next

Restart your node agent.

Restarting a node agent

Use the administrative tools to restart a node agent.

About this task

To restart a node agent, perform the following steps.

Procedure

1. From the administrative console of the deployment manager, click **System administration** > **Node agents**.
2. Select the node agent from the list on the Node Agent collection page.
3. Click **Restart**. The node agent is stopped and then restarted.

Starting and stopping deployment environments

You can start or stop deployment environments based on IBM-supplied patterns directly from the administrative console. You cannot manage custom deployment environments with this procedure.

Before you begin

- Verify that deployment environments exist on this deployment manager.
- Navigate to the administrative console of a deployment manager **Servers** > **Deployment Environments**.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

To start or stop a deployment environment, the deployment environment must exist.

About this task

Follow these steps when you want to start or stop a deployment environments based on IBM-supplied patterns.

Note: To start or stop a custom deployment environment, you must start and stop its clusters individually.

Note: Starting and Stopping deployment environment invokes a stop or start operation on all the clusters in the Deployment Environment at the same time, which might cause warnings or exceptions in the systemout.log file. If the deployment environment contains multiple clusters that depend on one another, it is recommended that you start and stop the individual clusters in the deployment environment as described in Starting a cluster and Stopping the cluster.

Procedure

1. Select the check boxes next to the names of the deployment environments to start or stop.
2. Take one of the actions:

Action	Result
Click **Start**.	The deployment manager starts the clusters that make up the deployment environments.
Click **Stop**.	The deployment manager stops the clusters that make up the deployment environments.

Note: This process can take several minutes depending on the size of your deployment environment.

Results

The display refreshes to indicate the status of the deployment environments.

Related reference:

"Cluster, single server and node status" on page 386
Use the indicators in the administrative console to see the status of specific entities within a deployment environment.

"Deployment environment function status" on page 387
Use the administrative console indicators to see the state of the minimum entities and the redundant entities in a configured deployment environment.

"Deployment environment status" on page 387
You can use the administrative console to see the state of a deployment environment.

Related information:

Using the Administration Thin Client

startDeploymentEnv command

stopDeploymentEnv command

Starting the deployment environment using the command line

You can start the deployment environment using the **wsadmin** command.

Before you begin

Ensure the **wsadmin** client can connect to the deployment manager for the deployment environment.

Required security role for this task: When security and role-based authorization are enabled, you must use a user ID and password with administrator or operator authority to perform this task.

About this task

To start the deployment environment with the **wsadmin** command, perform the following steps.

Procedure

1. Open a command window.
2. At the command prompt, enter the **wsadmin** command to enter the command environment. The **wsadmin** command is located in either the `<WPS>/profiles/<dmgr profile>/bin` directory or the `<WPS>/bin` directory.
3. Enter the **startDeploymentEnv** command to start the deployment environment.
4. If administrative security is on, enter your user ID and password when prompted.

Example

This example starts the deployment environment (**MyDepEnv**) on the host (**myDmgr**) with administrative security enabled.

Note: If you are running the **wsadmin** client from the deployment manager bin folder, you do not need to include the `-host` and `-port` parameters in the command.

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask startDeploymentEnv {-topologyName myDepEnv}
```

The `-connType` parameter specifies the type of connection to be used; the default argument is `SOAP`. If you are using a SOAP connection, including this parameter is optional.

The `-host` parameter specifies the host used for the SOAP or RMI connection. The default value for `-host` is the local host. If the node is running on the local host, you don not need to include this parameter.

If you disable administrative security, you do not need to provide a user ID and password.

Stopping the deployment environment using the command line

You can stop the deployment environment using the **wsadmin** command.

Before you begin

Ensure the **wsadmin** client can connect to the deployment manager for the deployment environment.

Required security role for this task: When security and role-based authorization are enabled, you must use a user ID and password with administrator or operator authority to perform this task.

About this task

To stop the deployment environment with the **wsadmin** command, perform the following steps.

Procedure

1. Open a command window.
2. At the command prompt, enter the **wsadmin** command to enter the command environment. The **wsadmin** command is located in either the `<WPS>/profiles/<dmgr profile>/bin` directory or the `<WPS>/bin` directory.
3. Enter the **stopDeploymentEnv** command to stop the deployment environment.
4. If administrative security is on, enter your user ID and password when prompted.

Example

This example stops the deployment environment (**MyDepEnv**) on the host (**myDmgr**) with administrative security enabled.

Note: If you are running the admin client from the deployment manager bin folder, you do not need to include the `-host` and `-port` parameters in the command.

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask stopDeploymentEnv {-topologyName myDepEnv}
```

The `-connType` parameter specifies the type of connection to be used; the default argument is `SOAP`. If you are using a SOAP connection, including this parameter is optional.

The `-host` parameter specifies the host used for the SOAP or RMI connection. The default value for `-host` is the local host. If the node is running on the local host, you don not need to include this parameter.

If you disable administrative security, you do not need to provide a user ID and password.

Starting a cluster

You can start all the servers in a cluster (cluster members) in one action. When you start a cluster you automatically enable workload management.

Before you begin

- Ensure that the node agents are running.
- Verify that all resources required by applications deployed to the cluster are available.
- Start all prerequisite subsystems.

About this task

When you request that all members of a cluster start, the cluster state changes to `partially started` and each server that is a member of that cluster launches, if it is not already running. After all members of the cluster are running, the cluster states changes to `running`.

The **Ripplestart** option first stops and then starts each server in turn.

Tip: If you are using clusters, the **Initial State** property of the Application Server subcomponent (**Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Administration** > **Server Components** > **Application Server**) is not intended to be used to control the state of individual servers in the cluster at the time the cluster is started. It is intended only as a way to control the state of the Application Server subcomponent of a server. It is best to start and stop the individual members of a cluster using the Server options of the administrative console or command line commands (**startServer** and **stopServer**).

If you use a deployment environment pattern of *Remote Messaging* or *Remote Messaging and Remote Support*, there can be multiple clusters that depend on one another. If such a case exists, start the infrastructure and the clusters as follows to avoid potential startup problems:

1. Infrastructure startup sequence:
   a. Database, Lightweight Directory Access Protocol (LDAP), and Web servers
   b. Deployment manager (if needed)
   c. Node agents
2. Cluster startup sequence:
   a. Messaging infrastructure cluster
   b. Support cluster (CEI)
   c. Application deployment cluster

To start a cluster, perform the following steps.

Procedure

1. From the administrative console of the deployment manager, click **Servers** > **Clusters** > **WebSphere application server clusters**.
2. Select the cluster you want to start.
3. Click **Start** or **Ripplestart** to start the cluster.

Note: If you use a deployment environment pattern of *Remote Messaging* or *Remote Messaging and Remote Support* make sure that you have started infrastructure components and that you start the clusters in the proper sequence, as described in About this task.

- **Start** launches the server process of each member of the cluster by calling the node agent for each server to start the servers. If your servers are stopped, select the **Start** option. If a call to a node agent for a server fails, the server does not start.
- **Ripplestart** combines stopping and starting operations. If your servers are running, select the **Ripplestart** option. It first stops and then restarts each member of the cluster. For example, your cluster contains 3 cluster members:
  - server_1
  - server_2
  - server_3

  When you click **Ripplestart**, server_1 stops and restarts, then server_2 stops and restarts, and finally server_3 stops and restarts.

  Use **Ripplestart** instead of manually stopping and then starting all of the application servers in the cluster.

  Note: The **Ripplestart** option restarts servers in sequence and ensures that at least one server in the cluster is online to handle requests.
  Attention: Do not perform a ripplestart on multiple clusters simultaneously. If plan on using **Ripplestart** to start clusters, do so on one cluster at a time.

Stopping a cluster

You can stop all the servers that are members of the same cluster at the same time by stopping the cluster.

Before you begin

Make sure there is no work in progress; performance monitoring infrastructure counters can indicate whether all queued work is complete. In addition, prevent new work from starting by disabling HTTP and IIOP traffic on the cluster members and quiescing the service integration buses.

About this task

If you use a deployment environment pattern of *Remote Messaging* or *Remote Messaging and Remote Support*, there can be multiple clusters that depend on one another. If such a case exists, stop the clusters in the following sequence:

1. Application deployment cluster
2. Support cluster (CEI)
3. Messaging infrastructure cluster

Procedure

1. From the administrative console of the deployment manager, click **Servers** > **Clusters** > **WebSphere application server clusters**.
2. Select the cluster you want to stop.
3. Click **Stop** or **Immediate Stop** to stop the cluster.
   - **Stop** halts each server in such a way that the server can finish work in progress. This option allows failover to another cluster member.

- **Immediate Stop** halts each server quickly, ignoring any current or waiting tasks.

Administering servers

Use the administrative interfaces to create, start, and stop servers. Servers extend the ability of application servers, allowing them to handle Service Component Architecture (SCA) modules. Other server processes, such as deployment managers and node agents, can be used to manage servers.

Servers must be running before you can start applications on them. The methods for starting a server vary and depend on whether you are starting a stand-alone server or a managed server. With managed servers, the node agent must be running before you can start the servers. You can start managed servers from the administrative console of the deployment manager. If you have deployment environments or clusters, you can start or stop all of the servers in one action, from the administrative console of the deployment manager.

Tip: If you are using clusters, the **Initial State** property of the Application Server subcomponent (**Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Administration** > **Server Components** > **Application Server**) is not intended to be used to control the state of individual servers in the cluster at the time the cluster is started. It is intended only as a way to control the state of the Application Server subcomponent of a server. It is best to start and stop the individual members of a cluster using the Server options of the administrative console or command line commands (**startServer** and **stopServer**).

Related concepts:

Servers
Servers provide the core functionality of WebSphere ESB. ESB servers extend, or augment, the ability of an application server to handle mediation modules. Other servers (deployment managers and node agents) are used for managing ESB servers.

Stand-alone server
A stand-alone server provides an environment for deploying mediation modules in one server process. This server process includes, but is not limited to, an administrative console, a deployment target, the messaging support, and a Common Event Infrastructure server.

Deployment managers
A deployment manager is a server that manages operations for a logical group, or cell, of other servers. The deployment manager is the central location for administering the servers and clusters.

Managed nodes
A *managed node* is a node that is federated to a deployment manager and contains a node agent and can contain managed servers. In a managed node, you can configure and run managed servers.

Related information:

Starting and stopping the deployment manager

Managing node agents

Starting stand-alone servers

Start a stand-alone application server in order to run the administrative console and any applications on the server.

About this task

To start a stand-alone server, perform the following steps.

Procedure

1. If you have created several stand-alone server profiles on the host, ensure that you are using the correct profile.
2. Start the server with one of the following methods.
   - Windows From the **Start** menu, click **IBM WebSphere** > **Enterprise Service Bus** > **Profiles** > *profile_name* > **Start the server**
   - In the First steps console, click **Start the server**.
   - Use the **startServer** command.
3. Verify that the server started successfully by checking the *install_root*/profiles/*profile_name*/logs/*server_name*/startServer.log log file for the message Server *server_name* open for e-business; process id is *nnnn*.

Related tasks:

Starting and stopping the administrative console
To access the administrative console, you must start it and then log in. After you finish working in the console, save your work and log out.

"Stopping stand-alone servers"
Stop a stand-alone server before modifying certain configurations for the server or the modules deployed on it.

Related information:

Starting the First steps console

startServer command

Stopping stand-alone servers

Stop a stand-alone server before modifying certain configurations for the server or the modules deployed on it.

Before you begin

Make sure that there is no work in progress; performance monitoring infrastructure counters can indicate whether all queued work is complete. In addition, prevent new work from starting by disabling HTTP and IIOP traffic and quiescing the service integration bus.

About this task

To stop a stand-alone server, perform the following steps.

Procedure

1. Stop the server with one of the following methods.
   - Windows From the **Start** menu, select **IBM WebSphere** > **Enterprise Service Bus** > **Profiles** > *profile_name* > **Stop the server**
   - In the First steps console, click **Stop the server**.
   - Use the **stopServer** command.
2. Verify that the server has stopped by checking the *install_root*/profiles/*profile_name*/logs/*server_name*/stopServer.log log file for the message Server *server_name* stop completed.

Related tasks:

"Starting stand-alone servers" on page 380
Start a stand-alone application server in order to run the administrative console and any applications on the server.

Related information:

Starting the First steps console

stopServer command

Starting managed servers

Start the application server process in order to run applications on the managed server.

Before you begin

Before you start a managed server, ensure that the following conditions are true:

- The deployment manager and node agent are running.
- All resources required by applications deployed to the server are available.
- All necessary subsystems are running.
- The server is not currently running.

About this task

If you are using clusters, the **Initial State** property of the Application Server subcomponent (**Servers** > **Application servers** > *server_name* > **Administration** > **Server Components** > **Application Server**) is not intended to be used to control the state of individual servers in the cluster at the time the cluster is started. It is intended only as a way to control the state of the Application Server subcomponent of a server. It is best to start and stop the individual members of a cluster using the Server options of the administrative console or command line commands (**startServer** and **stopServer**).

Important: If you are restarting a server after a failure and you want the recovery functions to finish before new work is started, do not use this procedure. Instead, restart the server in recovery mode using the **`startServer`** command with the **`-recovery`** option.

To start a managed server with the administrative console on the deployment manager, perform the following steps.

Procedure

1. Click **Servers** > **Server Types** > **WebSphere application servers**.
2. Select the server from the Application servers collection page.
3. Click **Start**.

What to do next

After you start the server you can start and manage the applications that run on the server. Note that it can take some time for all processes to discover a newly started server.

Related concepts:

"Servers and recovery mode processing" on page 1115
When you restart an application server instance with active transactions after a failure, the transaction service uses recovery logs to complete the recovery process.

Related tasks:

"Starting and stopping the administrative console" on page 365
To access the administrative console, you must start it and then log in. After you finish working in the console, save your work and log out.

"Stopping managed servers"
Stop the server before modifying certain configurations for the server or the modules deployed on it. Note that you can stop a single server in a cluster.

Related information:

startServer command

Enabling servers for Java debugging

Use the Java debugging mode to trace and debug a running server.

Before you begin

The server must be running.

About this task

To enable the Java debugging mode for a server, perform the following steps.

Procedure

1. From the administrative console of the deployment manager, click **Servers** > **Server Types** > **WebSphere application servers**, and then click the application server with the processes that you want to trace and debug..
2. Click **Server infrastructure** > **Java and Process Management** > **Process Definition** > **Java Virtual Machine**.
3. On the Java virtual machine page, click **Debug mode** to enable the standard Java debugger.
4. If required, set **Debug Mode** arguments.
5. Click **OK**.
6. Save the changes.
7. Stop and restart the server.

Stopping managed servers

Stop the server before modifying certain configurations for the server or the modules deployed on it. Note that you can stop a single server in a cluster.

Before you begin

Make sure that there is no work in progress; performance monitoring infrastructure counters can indicate whether all queued work is complete. In addition, prevent new work from starting by disabling HTTP and IIOP traffic and quiescing the service integration bus.

About this task

To stop a managed server, perform the following steps.

Procedure

1. From the administrative console on the deployment manager, click **Servers** > **Server Types** > **WebSphere application servers**.
2. Select the server from the Application servers collection page, and then click **Stop**.

Related tasks:

"Starting managed servers" on page 382
Start the application server process in order to run applications on the managed server.

Related information:

stopServer command

Administering deployment environments

Through the administrative console on the deployment manager you administer the deployment environments defined on the deployment manager. You can also create, delete, import, and export deployment environments from the administrative console.

Before you begin

Verify that a deployment manager is started and log in to the administrative console.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

About this task

Administer deployment environments when you need to update the deployment environments managed by a deployment manager. The **Deployment Environments** administrative console page is the starting point for all tasks related to the management and definition of deployment environments defined to a particular deployment manager.

Procedure

1. In the administrative console, click **Servers > Deployment Environments**.
2. To display the components of a deployment environment, click its name.
3. For existing environments, select the check box next to the deployment environments to manage and click one of the following buttons:

Function	Task
Start or **Stop**	Start and stop deployment environments
Remove	Remove resources from a deployment environment. This option does not delete the resources.
Export	Export deployment environments

4. To add new deployment environments to the deployment manager, use either **New** or **Import**.

What to do next

Manage deployment environment entities.

Modifying the deployment topology

Use the Deployment Topology page to manage the topology configuration for your IBM-supplied patterns. Managing the configuration can involve adding and replacing nodes, as well as changing the number of cluster members.

Before you begin

Navigate to the administrative console of a deployment manager **Servers** > **Deployment Environments** > *deployment_environment_name* > **Additional Properties** > **Deployment Topology**.

About this task

Use this page to add nodes to your deployment environment, if needed. You can also change the number of cluster members participating in a particular function for each node.

By adding nodes you can increase the overall work capacity of the system.

Use the procedure described in this task to modify the deployment topology in the following ways:

- Before the deployment environment is generated, so that only the deployment environment definition is updated
- After the deployment environment has been generated, as a means to create or delete cluster members

  If the deployment environment has already been generated, then the changes you make to the topology configuration are used to update the number of cluster members for the corresponding deployment environment clusters. For example, if you use the Deployment Topology panel to add a node, then the deployment environment definition is updated by that one node and all clusters that are managed by the deployment environment are adjusted by one more cluster member on that node (or the number of cluster members as defined by this panel). This adjustment is made to the topology configuration when you click **OK** or **Apply** on the Deployment Topology panel.

Procedure

- Select an objective and perform the associated actions.

Objective	Actions
Add node	Select a node from the pull-down list and click **Add**.
Change the number of cluster members involved in each function	Type the number in the entry field underneath the columns labeled: • Application Deployment Target • Messaging Infrastructure • Supporting Infrastructure **Remember:** You must have at least one cluster member assigned for each function.

Objective	Actions
Reset number of clusters assigned to each function	If you modified the number of clusters assigned to each cluster type, and you want to reset the parameters to the original value, click **Reset**
Remove a node from the deployment topology	For the node that you want to remove, select the check box in the Select column of the table and click **Remove**

- Click **Apply** to keep the updates and remain on the Deployment Topology page. Click **OK** to keep the updates and return to the previous page.

What to do next

Either save the changes or discard them.

Related concepts:

Custom deployment environment layout configuration
This overview describes two major configuration considerations for custom deployment environments: selecting clusters and single servers to use with the environment and specifying the deployment environment configuration. An understanding of these considerations enables you to plan and implement a deployment environment effectively.

Related reference:

"Cluster, single server and node status"
Use the indicators in the administrative console to see the status of specific entities within a deployment environment.

"Deployment environment function status" on page 387
Use the administrative console indicators to see the state of the minimum entities and the redundant entities in a configured deployment environment.

"Deployment environment status" on page 387
You can use the administrative console to see the state of a deployment environment.

Cluster, single server and node status

Use the indicators in the administrative console to see the status of specific entities within a deployment environment.

Table 23. Deployment environment status

Icon	Status	Description
	Unknown	The system cannot determine the status of the entity. This could indicate an incomplete configuration.
	Unavailable	The entity is configured, but unavailable.
	Stopped	The entity is stopped.
	Partially stopped	The entity is partially stopped. Less than all of the entities involved have stopped.
	Running	The entity is fully functional and running.
	Partially started	The entity is partially started. Less than all of the entities involved have started.

Deployment environment function status

Use the administrative console indicators to see the state of the minimum entities and the redundant entities in a configured deployment environment.

Explanation of entities state

The function status is typically used for clusters that perform a given function. A cluster is an example of a redundant entity state where its cluster members make up the redundant parts. A deployment environment status is an example of a minimum entity state where all functions need to be available for the deployment environment to be available.

Table 24. Aggregated state of entities

Icon	Status	Minimum entities state	Redundant entities state
	Unknown	At least one of the minimum entities state is unknown, making the entire state unknown.	The configuration did not complete for the deployment environment.
	Unavailable	At least one of the minimum entities is unavailable.	All of the entities in the deployment environment are unavailable.
	Stopped	All entities are stopped.	All minimum entities are stopped. If some entities are not stopped, this indicates those entities have problems.
	Partially stopped	There is at least one partially stopped entity and any number of stopped entities.	At least one partially stopped or stopped entity and any number of unavailable entities.
	Running	All entities are running.	All minimum entities are running. If some entities are not running, this indicates that those entities have problems.
	Partially started	There is at least one running entity and any number of stopped, partially stopped, or partially running entities.	At least one partially running or running and any number of stopped, partially stopped or unavailable entities.

Deployment environment status

You can use the administrative console to see the state of a deployment environment.

Table 25. States of a topology instance in order of least to most available

Status icon	Warning icon	State	Description
	None	Unknown	The system cannot determine the current state of the deployment environment.
		Incomplete	The deployment environment is not missing any elements but is incomplete in some way. The warning message contains additional details.
		Not configured	The configuration is known and complete but has not yet been generated.
		Deferred configuration	The deployment environment has been generated but deferred configuration has not been completed.
		Unavailable	The deployment environment is complete but at least one function is unavailable.
		Partially stopped	The deployment environment is available but at least one function is stopped or partially stopped.

Table 25. States of a topology instance in order of least to most available (continued)

Status icon	Warning icon	State	Description
		Stopped	All functions are stopped.
		Partially running	The deployment environment is available but at least one function is partially running.
		Running	The deployment environment is available and all functions are running.

Deleting deployment environment definitions using the command line

You can delete a deployment environment definition from a deployment manager using the **wsadmin** command. This will not impact any existing servers/clusters that are configured.

Before you begin

The admin client must connect to the deployment manager from which you are removing the deployment environment definition.

Verify that deployment environments exist on this deployment manager.

For recover purposes, consider exporting the deployment environment definition.

Required security role for this task: When security and role-based authorization are enabled, you must use a userid and password with administrator or operator authority to perform this task.

About this task

Delete the deployment environment definition from a deployment manager when you no longer need the specific definition.

This task uses **wsadmin** command to delete a deployment environment definition on the deployment manager.

You might want to use the command line to delete deployment environment definitions when you are making a large number of changes to a deployment environment. There is less overhead using the **wsadmin** command than there would be using the administrative console.

Procedure

1. Open a command window.

   The **wsadmin** command can be found at either the `<ESB>/profiles/<dmgr profile>/bin` directory, or the `<ESB>/bin` directory.
2. At the command prompt, enter the **wsadmin** command to enter the command environment.

   Note: Make sure **wsadmin** connects to the correct deployment manager, when running in connected mode.

3. Use the **deleteDeploymentEnvDef** command to delete the deployment environment definition from the deployment manager.

   Note: If administrative security is on, you will be prompted for a user ID and password, if you do not supply it in the command.

Example

This example deletes a deployment environment definition (**myDepEnv**) with administrative security enabled.

Note: If you are running the admin client from the deployment manager bin folder, you do not need to include the `-host` and `-port` parameters in the command.

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask deleteDeploymentEnvDef {-topologyName myDepEnv }
```

The `-connType` parameter specifies the type of connection to be used; the default argument is `SOAP`.

Note: As the default is `SOAP`, you do not need to give explicitly if SOAP is the connection type that is being used.

The `-host` parameter specifies the host used for the SOAP or RMI connection. The default value for `-host` is the local host.

Note: If the node is running on the local host, you don not need to specify `-host`

Note: If you disable administrative security, you do not need to provide a user ID and password.

To save this change to the master configuration issue a the command: **$AdminConfigSave**.

Related information:

Commands and scripts

deleteDeploymentEnvDef command

Renaming a deployment environment definition using the command line

You can rename a deployment environment definition using the **wsadmin** command.

Before you begin

You must be at the deployment manager from which you are renaming deployment environment definitions.

Required security role for this task: When security and role-based authorization are enabled, you must use a userid and password with administrator or operator authority to perform this task.

About this task

This task renames a deployment environment definition and uses the **wsadmin** command.

This command will fail if the deployment environment (topology) is already configured.

You would typically perform this task after importing a topology from another deployment environment definition. There is less overhead using the **wsadmin** command than there would be using the administrative console.

Procedure

1. Open a command window. .

   The **wsadmin** command can be found at either the `<ESB>/profiles/<dmgr profile>/bin` directory, or the `<ESB>/bin` directory.
2. At the command prompt, enter the **wsadmin** command to enter the **wsadmin** environment.
3. Use the **renameDeploymentEnvDef** command to rename a deployment environment definition.

   Note: If administrative security is on, you will be prompted for a user ID and password, if you do not supply it in the command.

Example

This example renames a deployment environment definition (**TheOldDepEnvName**) to the (**TheNewDepEnvName**) with administrative security enabled:

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgrAdmin -password -dmgrPass
> $AdminTask renameDeploymentEnvDef {-topologyName myDepEnv
-oldName TheOldDepEnvName -newName TheNewDepEnvName}
```

The `-connType` parameter specifies the type of connection to be used; the default argument is `SOAP`.

Note: As the default is `SOAP`, you do not need to give explicitly if SOAP is the connection type that is being used.

The `-host` parameter specifies the host used for the SOAP or RMI connection. The default value for `-host` is the local host.

Note: If the node is running on the local host, you don not need to specify `-host`

Note: If you disable administrative security, you do not need to provide a user ID and password.

Related information:

Commands and scripts

renameDeploymentEnvDef command

Removing nodes from a deployment environment definition using the command line

You can remove nodes from a deployment environment definition using the **wsadmin** command.

Before you begin

This command to remove a node from the deployment environment will fail if the topology is already configured.

The admin client must connect to the deployment manager from which you are removing the node.

Required security role for this task: When security and role-based authorization are enabled, you must use a userid and password with administrator or operator authority to perform this task.

About this task

This task uses the **wsadmin** command to remove a node from a deployment environment definition.

You might want to use the command line to remove a federated node from a deployment environment when you are making a large number of changes to a deployment environment. There is less overhead using the **wsadmin** command than there would be using the administrative console.

Procedure

1. Open a command window.

   The **wsadmin** command can be found at either the `<ESB>/profiles/<dmgr profile>/bin` directory, or the `<ESB>/bin` directory.
2. At the command prompt, enter the **wsadmin** command to enter the command environment.

   Note: Make sure **wsadmin** connects to the correct deployment manager, when running in connected mode.
3. Use the **removeNodeFromDeploymentEnvDef** command to remove the node from the deployment environment definition.

   Note: If administrative security is on, you will be prompted for a user ID and password, if you do not supply it in the command.

Example

This example removes a node (**MyNode**) from a Messaging cluster (**Messaging**) for the deployment environment definition (**myDepEnv**) with administrative security enabled.

Note: If you are running the admin client from the deployment manager bin folder, you do not need to include the `-host` and `-port` parameters in the command.

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgrAdmin -password -dmgrPass
> $AdminTask removeNodeFromDeploymentEnvDef -topologyName myDepEnv
-topologyRole Messaging -nodeName MyNode
```

The `-conntype` parameter specifies the type of connection to be used; the default argument is `SOAP`.

Note: As the default is `SOAP`, you do not need to give explicitly if SOAP is the connection type that is being used.

The `-host` parameter specifies the host used for the SOAP or RMI connection. The default value for `-host` is the local host.

Note: If the node is running on the local host, you don not need to specify `-host`

Note: If you do not specify a value for **topologyRole**, the node is removed from every role (cluster) in the environment definition.

Note: If you disable administrative security, you do not need to provide a user ID and password.

To save this change to the master configuration issue the command: **$AdminConfig Save**

Related information:

Commands and scripts

removeNodeFromDeploymentEnvDef command

Renaming nodes in a deployment environment definition using the command line

You can rename nodes in a deployment environment definition using the **wsadmin** command.

Before you begin

The admin client has to connect to the deployment manager from which you are renaming nodes in the deployment environment definition.

Required security role for this task: When security and role-based authorization are enabled, you must use a userid and password with administrator or operator authority to perform this task.

About this task

This task renames a node in deployment environment definition and uses the **wsadmin** command.

This command will fail if the deployment environment (topology) is already configured.

You would typically perform this task after importing a deployment environment definition. There is less overhead using the **wsadmin** command than there would be using the administrative console.

Procedure

1. Open a command window.

   The wsadmin command can be found at either the `<ESB>/profiles/<dmgr profile>/bin` directory, or the `<ESB>/bin` directory.
2. At the command prompt, enter the **wsadmin** command to enter the command environment.

   Note: Make sure **wsadmin** connects to the correct deployment manager, when running in connected mode.

3. Use the **renameNodeInDeploymentEnvDef** command to rename a node in the deployment environment definition.

   Note: If administrative security is on, you will be prompted for a user ID and password, if you do not supply it in the command.

Example

This example renames a node (**TheOldNodeName**) to the (**TheNewNodeName**) for the deployment environment definition (**myDepEnv**) with administrative security enabled:

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgrAdmin -password -dmgrPass
> $AdminTask renameNodeInDeploymentEnvDef -topologyName myDepEnv
-oldName TheOldNodeName -newName TheNewNodeName
```

The `-connType` parameter specifies the type of connection to be used; the default argument is `SOAP`.

Note: As the default is `SOAP`, you do not need to give explicitly if SOAP is the connection type that is being used.

The `-host` parameter specifies the host used for the SOAP or RMI connection. The default value for `-host` is the local host.

Note: If the node is running on the local host, you don not need to specify `-host`

Note: If you disable administrative security, you do not need to provide a user ID and password.

To save this change to the master configuration issue the command: **$AdminConfig Save**.

Related information:

Commands and scripts

renameNodeInDeploymentEnvDef command

Modifying deployment environment definition parameters

You can use the AdminConfig object to modify parameters in the deployment environment definition.

Before you begin

AdminConfig communicates with the configuration service component to make configuration inquires and changes. You can use it to query existing configuration objects, create configuration objects, modify existing objects, remove configuration objects, and obtain help.

The admin client has to connect to the deployment manager from which you are changing parameters for the deployment environment definition.

Required security role for this task: When security and role-based authorization are enabled, you must use a userid and password with administrator or operator authority to perform this task.

About this task

When you create a deployment environment definition, the admin task selects default parameters based on the common database (CommonDB) selected when you created the Deployment Manager.

Procedure

1. Use **AdminConfig** to modify any property in the deployment environment definition.

   The following list provides a general method to update configuration objects:
   - Identify the configuration type and the corresponding attributes.
   - Query an existing configuration object to obtain a configuration ID to use.
   - Modify the existing configuration object or create a new one.
   - Save the configuration.
2. Save the configuration changes. To save this change to the master configuration issue the command: **$AdminConfig Save**

Related information:

Commands and scripts

Using the AdminConfig object for scripted administration

Managing deployment environment resources

You can manage the resources of your deployment environment to address changing requirements over time.

Before you begin

- Verify that the deployment environments exist on this deployment manager.
- On the administrative console of the deployment manager, click **Servers** > **Deployment Environments**.
- You must completely stop any nodes you are removing from the deployment environment before removing those nodes.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

About this task

As your deployment environment needs evolve, you can manage its resources to address new demands and processing requirements.

In managing the resources of a deployment environment, you can do the following:

- Add or remove servers and clusters.
- Change which nodes participate in specific functions.
- Change the configuration of data sources.
- Change authentication aliases.
- Obtain information on how to configure databases or tables if you previously deferred that operation.

Procedure

1. Select the deployment environment for which you want to manage resources by clicking the name. The system displays the Deployment Environment Configuration page which lists:
   - **Deployment Environment**
   - **Deployment Environment Pattern**
   - **Description**
   - **Deployment Environment Status**
   - **Deployment Environment Functions**
   - Links to the configuration pages
2. Select the configuration area of the deployment environment to manage. Select each link until you complete your changes.

Configuration area	Available actions
Additional Properties	**Deployment Topology** To change the configuration of a deployment environment based on IBM-supplied patterns. **Deferred Configuration** To determine any manual steps needed to complete the configuration of this deployment environment.
Related Items	**Data Sources** To change the data source configuration for the various components within the deployment environment. **Authentication Aliases** To change the authentication alias or password for components within the deployment environment.

3. Complete the configuration by choosing the option for the result needed.

   Note: The system does not complete the configuration until you click **Generate Environment**.

Action	Result
Click **OK** or **Apply**	Both options save the configuration. **Apply** leaves you on the current page, **OK** returns you to the Deployment Environments page.
Click **Generate Environment**	Saves the configuration and starts the configuration process. **Note:** If the deployment environment does not meet the minimum constraints or is incomplete, you cannot select this option.

What to do next

Manage the deployment environment.

Editing the data source configuration

After you create a deployment environment, you can edit the data source configuration. The Data Sources page lists all of the data sources in your deployment environment, and you can perform multiple edits on this page.

Before you begin

- Verify that deployment environments exist on this deployment manager.
- Navigate to the administrative console of a deployment manager **Servers** > **Deployment Environments** > *deployment_environment_name* > **Related Items** > **Data Sources**.

Required security role for this task: When security and role-based authorization are enabled, you must be logged in to the administrative console as an administrator or a configurator to perform this task.

About this task

The Data Sources page allows you to edit the collection of all the data sources in your deployment environment. Although you can edit data sources on this page, you cannot add a new data source here. The number of text fields might differ depending on the component and data source provider for each data source.

Important: If you make edits that conflict, such as using a schema name that is used by another data source, the system displays a warning message. You can save your changes, but the message persists until you resolve the conflict.

Procedure

1. On the Data Sources page, select the component that contains the data source to edit.
2. Make any required changes.
3. Click **Apply** or **OK** to save the changes.

Editing your database provider

Use the Database Source Provider Configuration page to make changes to your database provider.

Before you begin

- Verify that deployment environments exist on this deployment manager.
- Navigate to the administrative console of a deployment manager **Servers** > **Deployment Environments** > *deployment_environment_name* > **Related Items** > **Data Sources**.

Required security role for this task: When security and role-based authorization are enabled, you must be logged in to the administrative console as an administrator or a configurator to perform this task.

About this task

Use this procedure when you need to make changes to the configuration of a database provider used by a data source. Some sections of the Database Provider Configuration page, such as **Component Specific Properties**, have a different number of text boxes depending on the database provider.

Procedure

1. On the Data Sources page, select a data source for the data source provider you want to edit.
2. Click **Edit Provider** to open the Database Source Provider Configuration page.
3. Make any required changes.
4. Click **Apply** or **OK** to save your changes.

Related information:

Configuring databases

Common database specifications

Editing a data source in your deployment environment

Use the Data Source page to edit your data source properties.

Before you begin

- Verify that deployment environments exist on this deployment manager.
- Navigate to the administrative console of a deployment manager **Servers** > **Deployment Environments** > *deployment_environment_name* > **Related Items** > **Data Sources**.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

About this task

Use this procedure to update the data sources used by a deployment environment through the Data Sources page. You cannot add a new data source in the Data Sources page. You can edit data source information by either clicking the data source name or selecting the component. Some text boxes are unavailable and you cannot change these values.

Important: If you make edits that conflict, such as using a schema name that is used by another data source, the system displays a warning message. You can save your changes, but the message persists until the conflict is resolved.

Procedure

1. On the Data Sources page, select the component that contains the data source you want to change and click **Edit**.
2. Edit the relevant information.
3. Click **Apply** or **OK** to save your changes.

Stopping and restarting the deployment manager

After any configuration changes to the deployment manager, you must stop and restart the deployment manager before those changes take effect.

Before you begin

Verify that a deployment manager is started and log in to the administrative console.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

Procedure

1. Choose a method to stop the deployment manager.

Method	Actions
Using the administrative console	1. Navigate to **System Administration** > **Deployment Manager**. 2. Click **Stop**.
Using the command line	1. Navigate to the deployment manager *profile_root*/bin directory. 2. Enter the **stopManager** command for your operating system. **Note:** If administrative security is enabled, the system prompts you to enter a user ID and password.

2. Wait for verification that the deployment manager has stopped.
3. Navigate to the deployment manager *profile_root*/bin directory.
4. Enter the **startManager** command for your operating system.

   Note: If administrative security is enabled, the system prompts you to enter a user ID and password.

What to do next

Verify that the application deployment target cluster can start.

Stopping and restarting a cluster member

When you make configuration changes, you must stop and restart a cluster member.

Before you begin

1. Prevent new work from entering the cluster member:
   - If you are using the IBM HTTP Server, change the plugin_cfg.xml file to remove the cluster member for HTTP traffic. If you are using another HTTP server, follow the directions for your HTTP server to remove the cluster member.
   - For IIOP traffic, set the runtime weight to zero for the cluster member.
2. Verify that work that is destined for the cluster member is complete. Either wait a period of time or use Performance Monitoring Infrastructure counters to determine when the cluster completes all of the queued work.

About this task

Some configuration changes require you to stop and restart server processes before the configuration change takes effect. This involves stopping and restarting of the deployment manager, cluster member, and node agent.

Note: All command files (alternatives to using the administrative console) are located in the *install_root*/bin subdirectory.

To stop and restart a server using the administrative console complete the following steps:

Procedure

1. In the administrative console, navigate to **Servers** > **Server Types** > **WebSphere application servers**.
2. Select the servers or cluster members to be stopped and click **Stop**.
3. Wait for the servers or cluster members to stop.
4. Select the servers or cluster members to be restarted and click **Start**.
5. Wait for the servers or cluster members to start.

   Note: Alternatively, you can stop and restart cluster members from the command line using the **`stopServer`** and **`startServer`** commands for your operating system or from the administrative console cluster panel by selecting **Servers** > **Clusters** > **WebSphere application server clusters** > ***cluster_name***.

Starting and stopping deployment environments

You can start or stop deployment environments based on IBM-supplied patterns directly from the administrative console. You cannot manage custom deployment environments with this procedure.

Before you begin

- Verify that deployment environments exist on this deployment manager.
- Navigate to the administrative console of a deployment manager **Servers** > **Deployment Environments**.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

To start or stop a deployment environment, the deployment environment must exist.

About this task

Follow these steps when you want to start or stop a deployment environments based on IBM-supplied patterns.

Note: To start or stop a custom deployment environment, you must start and stop its clusters individually.

Note: Starting and Stopping deployment environment invokes a stop or start operation on all the clusters in the Deployment Environment at the same time, which might cause warnings or exceptions in the systemout.log file. If the deployment environment contains multiple clusters that depend on one another, it is recommended that you start and stop the individual clusters in the deployment environment as described in Starting a cluster and Stopping the cluster.

Procedure

1. Select the check boxes next to the names of the deployment environments to start or stop.
2. Take one of the actions:

Action	Result
Click **Start**.	The deployment manager starts the clusters that make up the deployment environments.

Action	Result
Click **Stop**.	The deployment manager stops the clusters that make up the deployment environments.

Note: This process can take several minutes depending on the size of your deployment environment.

Results

The display refreshes to indicate the status of the deployment environments.

Related reference:

"Cluster, single server and node status" on page 386
Use the indicators in the administrative console to see the status of specific entities within a deployment environment.

"Deployment environment function status" on page 387
Use the administrative console indicators to see the state of the minimum entities and the redundant entities in a configured deployment environment.

"Deployment environment status" on page 387
You can use the administrative console to see the state of a deployment environment.

Related information:

Using the Administration Thin Client

startDeploymentEnv command

stopDeploymentEnv command

Exporting deployment environment definitions using the administrative console

Exporting deployment environment definitions helps you speed the implementation of deployment environments by minimizing the configuration on each deployment manager. You can use the exported deployment environment on other deployment managers as a template for the deployment environment. You can also replicate the same deployment environment configuration on a large scale.

Before you begin

- Define at least one deployment environment on a deployment manager.
- Log in to the administrative console of the deployment manager from which you are exporting the deployment environment definitions.
- On the administrative console of the deployment manager navigate to **Servers** > **Deployment Environments**.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

About this task

If you are implementing a number of deployment environments based on the same designs, you can export the deployment environment definitions to use as templates for deployment environments on other deployment managers.

Procedure

1. On the Deployment Environments page, select the check boxes next to the deployment environment definitions to export.
2. Click **Export**. The system response depends on whether you select:

Number of deployment environments to export	Action
One	At the prompt, enter the name of the exported file. The default name is *deployment_environment_name*.xml. To change the default name, specify the full file path.
Multiple	At the prompt, enter the output directory in which to place the exported compressed file that contains the deployment environment definitions. By default, the system names the compressed file *first_env_name*.zip. To change the default name, specify the full file path. **Note:** You cannot directly import a compressed file, you must extract the deployment environment definitions to the target file system.

3. Verify that the system created the files.

What to do next

You can import the exported files to other deployment managers.

Exporting deployment environment definitions using the command line

You can export deployment environment definitions using the **wsadmin** command. You can use the **wsadmin** command to perform the same definition-export task that you perform in the administrative console. This capability allows you to use a script to export large numbers of deployment environment definitions from a deployment manager freeing the administrative console for other tasks and enables you to replicate working configurations to other deployment managers.

Before you begin

You must be at the deployment manager from which you are exporting the deployment environment definitions.

Required security role for this task: When security and role-based authorization are enabled, you must use a userid and password with administrator or operator authority to perform this task.

About this task

You can use the command line to export deployment environment definitions in the following situations:

- You must export multiple deployment environment definitions and prefer to use the command line.

- You prefer to use the command line to export one deployment environment definition.
- You must export a large number of deployment environment definitions; using **wsadmin** reduces the time for performing the task.

Procedure

1. Open a command window.

   The **wsadmin** command can be found at either the `<ESB>/profiles/<dmgr profile>/bin` directory, or the `<ESB>/bin` directory.
2. At the command prompt, enter the **wsadmin** command to enter the **wsadmin** environment.
3. Use the **exportDeploymentEnvDef** command to export the deployment environment definition from the deployment manager to an output file. The file name is in the form *depEnvName*`.xml`

   Note: If administrative security is on, you will be prompted for a user ID and password, if you do not supply it in the command.

Example

This example exports the deployment environment **myDepEnv** on the host **myDmgr** with administrative security enabled.

Note: If you are running the admin client from the deployment manager bin folder, you do not need to include the `-host` and `-port` parameters in the command.

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask exportDeploymentEnvDef {-filePath c:/dmgr01/DeploymentEnvs
-topologyName myDepEnv}
```

The `-connType` parameter specifies the type of connection to be used; the default argument is `SOAP`.

Note: As the default is `SOAP`, you do not need to give explicitly if SOAP is the connection type that is being used.

The `-host` parameter specifies the host used for the SOAP or RMI connection. The default value for `-host` is the local host.

Note: If the node is running on the local host, you don not need to specify `-host`

Note: If you disable administrative security, you do not need to provide a user ID and password.

Related information:

exportDeploymentEnvDef command

Importing deployment environment definitions based on design documents using the administrative console

You can import an existing deployment environment definition based on a design document from another deployment manager to use as a base for configuring a new deployment environment.

Before you begin

- On the administrative console of the deployment manager navigate to **Servers** > **Deployment Environments**.
- You must have a copy of an exported deployment environment design document from another deployment manager.
- You must be able to access the deployment environment design document (an XML file) from the deployment manager into which you are importing the deployment environment design.
- The deployment manager that imports the deployment environment definition must support at least all of the functions that are defined in the deployment environment design document.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

Important: You cannot import multiple deployment environment design documents from a compressed file at the same time. You must extract the design documents from the compressed file and then import the XML files one at a time.

About this task

Importing an existing deployment environment design to create a new one can minimize the amount of time you spend configuring a deployment environment. If an existing environment is similar to the one you want to create, export it and then import it into the deployment manager you are configuring.

Procedure

1. Click **Import** in the Deployment Environments page to launch the Deployment Environment Configuration wizard.

   The wizard starts with **Create a deployment environment based on an imported design** selected.
2. Click **Browse** to open a file dialog and select the deployment environment design document (XML file) to import or type the full path to it.
3. Click **Next** to load the configuration and launch the Import deployment environment wizard.

   The wizard displays the Select Nodes page.
4. Optional: From the list of possible nodes on the Select Nodes page, select the nodes to include in the deployment environment and click **Next**.

   To include a node, select the check box next to the node name. Use **Node Mapping** to map the selected node to another node name.

   Important: **Next** is not available if the nodes selected do not meet the constraints imposed by the imported deployment environment design. For example, if there is a requirement for the deployment environment to contain a node named “Mandatory_Node” and 3 other nodes with any name, you will be unable to continue until you select “Mandatory_Node” and 3 other nodes.
5. Optional: On the Clusters page, assign the required number of cluster members on each node for each cluster *type* (Application Deployment Target, Messaging Infrastructure and Supporting Infrastructure) of the deployment environment.

   By default one cluster member is assigned on each node for each function. You change the number by replacing the number in each column. If you are

unfamiliar with the different cluster roles and functions provided by each type of cluster, see "Topology types and deployment environment patterns."

A 0 (zero) value for a node means that the node does not contribute to the selected function, based on features that you have selected.

After assigning cluster members, you can click **Next** to display the Cluster naming pages for each cluster type of the deployment environment. The Cluster naming sub-steps that display will vary depending on the deployment environment pattern selected.

The system generates default values for cluster names and cluster member names.

If you do not want to customize cluster names or cluster member names, you can use the wizard navigation pane to go directly to the REST Services page in a following step.

Each substep page is structured in the same fashion, and is described in Customize the cluster names and cluster member names.

a. Optional: Customize the cluster names and cluster member names.

   Use the Cluster Naming page to customize cluster names or cluster member names for the cluster type. There is one substep page for each cluster *type* in the pattern that you have selected. For example, if you selected a **Remote messaging and remote support pattern**, there are 3 sub-steps, one for each type of cluster (Application Deployment Target, Messaging Infrastructure and Supporting Infrastructure) in that pattern.

   The information on each substep page is as follows:

   Cluster

   A read-only field specifying the functional role of the cluster.

   The value varies depending on the cluster type, as follows:

   - Application Deployment Target
   - Messaging Infrastructure
   - Supporting Infrastructure
   - Web Application Infrastructure

   For information on the functional role provided by each cluster type, see Topology types and deployment environment patterns

   Cluster Name

   Contains the system-generated default value for the cluster name.

   The default values are based on a naming convention of `<Deployment Environment Name>.<Cluster type name>`, where`cluster type name` is one of the following values:

   - AppTarget

     For clusters performing the role of application deployment target
   - Messaging

     For clusters performing the role of messaging infrastructure
   - Support

     For clusters performing the role of supporting infrastructure
   - Web

     For clusters performing the role of supporting web applications.

Note: This cluster type name applies for BPM configurations in which WebSphere Business Monitor is the primary feature / product.

Cluster Member Name

Accept the system-generated default value or specify a name of your choosing.

The default value for the cluster member name is based on the following naming convention: `<cluster name>.<node name>.<node number sequence> .`

The number of cluster member names that display in the table match the number of cluster members that you entered for the cluster type column and node row on the Clusters page. See the preceding step for the Clusters page.

6. On the REST Services page, configure service endpoints for Representational State Transfer (REST) application programming interfaces (APIs).

   If you want widgets to be available in Business Space, you must configure the REST service endpoints for those widgets.

   a. Configure a full URL path for all REST services by selecting either **https://** or **http://** from the **Protocol** list.
   b. Enter a name in the **Host Name or Virtual Host in a Load-Balanced Environment** field.
   c. In the **Port** field, enter the port that a client needs to communicate with the server or cluster.
   d. In the table of REST services, if you want to modify the description of the REST service endpoint, overtype the entry in the Description field. The other fields are read-only.
   e. Click **Next** to go to the Import the database configuration page.

7. Optional: On the Import the database configuration page, click **Browse** to go the database design document or enter the path to the database design document and then click **Next** to go to the Data sources page. The design document can be based on a database design that you created using the database design tool (DDT), or it can be the supplied design document based on the pattern and feature that you have selected.

   Note: The database design document that you import for the deployment environment does not change the commonDB created at Profile Creation time.

8. Required: On the Database page, configure the database parameters for data sources of the deployment environment, then click **Next** to go to the Security page.

   Note: The database specified in this panel must already exist. Deployment environment configuration never creates a new database.

   On this page, define the database information for the components that are included in this deployment environment. Where possible, the wizard supplies default information for the parameters, but change those values to match the values that you defined when you planned the environment.

   Note: If you imported a database design document, the information on the Database page reflects the data source configuration as it exists in the database design document that you imported.

Whether or not this step displays for a fast path deployment environment configuration is conditional. This step displays for a fast path deployment environment configuration if more than one database has been defined.

This step always displays if you are using DB2 for z/OS or an Oracle database provider.

Note: The default schema names that are displayed on this page might conflict with your site naming convention or might conflict with existing schemas. As such, it is likely that you will need to change the schema name. Pay close attention to the values specified to avoid potential naming conflicts.

Oracle database considerations:

- Make sure that the username and schema name are exactly the same. The user specified should exist in the database before generating the environment.

SQL Server considerations:

- Make sure that the username and schema exist before the configuration is done. The schema value should be the default schema for the user chosen.
- To indicate that users will connect to the databases using Windows credentials, select the individual data source, click **Edit**, and select **Apply Windows authentication**.

For a production environment, you should set the same values for **User name** and **Schema name** and you should deselect **Create tables**. For a production environment, create the required schemas manually and use the SQL files generated to create the tables.

Note: You cannot select **Create tables** for Business Space (the option is unavailable for selection). The SQL files for Business Space need to be run manually. For information on running the SQL manually for Business Space, see Configuring Business Space database tables.

You can edit all key parameters, such as the database name, whether or not to create tables, the data source runtime user name, and the password for the deployment environment.

You can select which database to use for the given component.

DB2 for z/OS: The **Create tables** option cannot be used if you are using a DB2 for z/OS database provider.

Steps that cannot be completed through the Deployment Environment Configuration wizard, and which need to be completed manually, are listed on the Deferred Configuration page.

9. On the Security page, configure the authentication aliases WebSphere uses when accessing secure components

   You can change the authentication alias user name and password on this page. These aliases are used to access secure components but do not provide access to data sources

10. Optional: On the System web applications page, set the context root for component-based web applications in your deployment environment or accept the system-provided default values for the context roots. Then click **Next** to display the Summary page.

    The System web applications page displays for deployment environments using the Remote messaging, support and web applications pattern.

    The table contains the following control information.

Web Application
The name of the Web application.

Some of the components that are part of the deployment environment you are creating contain web applications. The **Web application** column can include the following components:

- Business Space

Context Root
The current value of the context root for the component.

By default, the default context root for the web application applies. You can change the context roots by typing over the value in the **Context Root** field.

Note: The Business Space context root is read only and cannot be edited.

11. Verify that the information on the Summary page is correct and perform the following substeps:
    a. Optional: If you do not want to save the deployment environment configuration, you can click **Cancel**.
    b. Optional: If you want to exit without generating the configuration, click **Finish**.

       To get back to the panel (if you exited without completing), perform the following from the administrative console: **Deployment Environments** > *name of deployment environment* > **Generate Environment** .
    c. To save the deployment environment configuration, click **Finish** and from within the Messages window, click **Save**.

       Clicking **Save** saves the deployment environment to the master configuration. If an error occurs during deployment environment generation, the configuration settings are saved to the master configuration.
    d. Check for deferred configuration steps

       Select **Deployment Environments** > *name of deployment environment* > **Deferred Configuration**

       You need to address any existing deferred configuration steps before starting the Deployment Environment.
    e. If you are satisfied with the deployment environment configuration and you have addressed any of the deferred configuration steps, click **Finish and Generate Environment** to save and complete the configuration of the deployment environment.

Results

When the configuration completes, you can examine the configuration files to view the changes.

What to do next

Either save the changes to the master configuration or discard them.

Related tasks:

"Exporting deployment environment definitions using the administrative console" on page 400
Exporting deployment environment definitions helps you speed the implementation of deployment environments by minimizing the configuration on each deployment manager. You can use the exported deployment environment on other deployment managers as a template for the deployment environment. You can also replicate the same deployment environment configuration on a large scale.

Importing deployment environment definitions using the command line

You can import deployment environment definitions using the **wsadmin** command. You can use the **wsadmin** command to perform the same definition-import task that you perform in the administrative console. This capability allows you to use a script to import many deployment environment definitions to a deployment manager freeing the administrative console for other tasks and enables you to replicate working configurations to other deployment managers.

Before you begin

- You must have a copy of the exported deployment environment definition.
- You must be at the deployment manager to which you are importing the deployment environment definition.
- Make sure that a deployment environment with the same name as the deployment environment definition you are importing does not exist on this deployment manager.
- The deployment manager from which you are importing the deployment environment definition must at least support all the functions defined in the deployment environment design.

Required security role for this task: When security and role-based authorization are enabled, you must use a userid and password with administrator or operator authority to perform this task.

About this task

Use the command line to import deployment environment definitions in the following situations:

- You must import multiple deployment environment definitions and prefer to use the command line.
- You prefer to use the command line to import one deployment environment definition as a template for multiple deployment environments.
- You must import a large number of deployment environment designs; using **wsadmin** reduces the time for performing the task.

Procedure

1. Open a command window.

   The **wsadmin** command can be found at either the `<ESB>/profiles/<dmgr profile>/bin` directory, or the `<ESB>/bin` directory.
2. Copy the deployment environment definition XML file you are importing to the system.
3. Enter the **wsadmin** command to enter the **wsadmin** environment.

4. Use the **importDeploymentEnvDef** command to import the deployment environment definition from the file you just copied to the deployment manager. You can rename the deployment environment when you import it.

   Note: If administrative security is on, you will be prompted for a user ID and password, if you do not supply it in the command.

Example

This example imports the deployment environment myDepEnv and renames it eastDepEnv on the deployment manager myDmgr with administrative security enabled.

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgrAdmin -password -dmgrPass
> $AdminTask importDeploymentEnvDef {-filePath
c:/dmgr01/importedEnvironments/myDepEnv.xml -topologyName eastDepEnv}
```

The `-connType` parameter specifies the type of connection to be used; the default argument is `SOAP`.

Note: As the default is `SOAP`, you do not need to give explicitly if SOAP is the connection type that is being used.

The `-host` parameter specifies the host used for the SOAP or RMI connection. The default value for `-host` is the local host.

Note: If the node is running on the local host, you do not need to specify `-host`

Note: If you disable administrative security, you do not need to specify a user ID and password.

What to do next

Optiional: Validate the imported deployment environments.

Related information:

Generating deployment environments using the command line

Managing node agents

importDeploymentEnvDef command

Removing deployment environments

Removing a deployment environment removes the management entity of the deployment environment. Deleting the deployment environment does not remove or change the configuration of the servers, nodes, and clusters that make up the deployment environment. Deleting deployment environments might be the final phase of moving a deployment environment from one deployment manager to another.

Before you begin

- On the administrative console of the deployment manager navigate to **Servers** > **Deployment Environments**.
- Verify that deployment environments exist on this deployment manager.
- For recover purposes, consider exporting the deployment environment definition.

Required security role for this task: When security and role-based authorization are enabled, you must log in to the administrative console as an administrator or configurator to perform this task.

About this task

When you no longer need to manage the resources of a specific deployment environment as a group, remove that deployment environment definition from the deployment manager.

Procedure

1. On the Deployment Environment page, select the check box for the deployment environments to remove and click **Remove**.

   The system removes the deployment environment from the display.
2. Click **Save** to save this change to the master configuration or **Discard** to prevent the update of the master configuration.

Related tasks:

"Exporting deployment environment definitions using the administrative console" on page 400
Exporting deployment environment definitions helps you speed the implementation of deployment environments by minimizing the configuration on each deployment manager. You can use the exported deployment environment on other deployment managers as a template for the deployment environment. You can also replicate the same deployment environment configuration on a large scale.

Administering applications and application services

Applications for WebSphere ESB involve similar administration tasks and interfaces as Java EE applications for WebSphere Application Server, with some additional tasks specifically relating to service applications, service modules, WebSphere MQ destinations, and other resources.

Related information:

Deploying and administering enterprise applications

Administering service applications and service modules

Use the administrative tools to view and manage service applications and their associated service modules.

Before you begin

Deploy your service modules to the runtime environment.

About this task

A service module is a Service Component Architecture (SCA) module that provides services in the run time. When you deploy a service module to WebSphere ESB, you build an associated service application that is packaged as an Enterprise Archive (EAR) file.

Versioning in service applications

Service applications support versioning. You can develop and deploy one or more versions of a module and its artifacts into the runtime environment for use by specific clients.

Note: This topic describes how versioning works in a service application deployed directly from Integration Designer. See the related topic for information on the versioning of process applications deployed from the repository in Process Center.

What can be versioned?

A module can have a version number, as can the SCA import and export bindings in a module. SCA bindings inherit their version information from the module they are associated with.

Note: At this time, SCA bindings are the only binding type that can be versioned. Versioning is optional for modules. Modules developed and deployed with versions of WebSphere Integration Developer and WebSphere Process Server before 7.0 do not have versions and continue to function with their current behavior. Refer to the migration topics for more information.

Libraries can also be versioned. Modules that use a library have a dependency on a specific version of that library, and libraries can also have dependencies on specific versions of other libraries.

Considerations for deploying versioned modules

You can deploy a versioned module into the run time and administer it from the SCA Modules pages within the administrative console. WebSphere ESB supports the following versioned deployment scenarios:

- Installing a versioned module to a server or cluster in a cell
- Installing the same version of a module once to each of one or more servers or clusters in a cell
- Installing different versions of a module on the same server or cluster

Deploying a new version of a module does not replace any previous versions of the module. Previous versions of cell-scoped application artifacts (in this case, business rules) are overwritten.

If you want to update an application (for example, to make minor corrections or improvements) without changing the version, that updated application and its artifacts will replace the existing application and artifacts, with the exception of any defined security policies. All security policy artifacts are preserved during an application update.

To preserve versioning information, the installation process automatically changes the module name (by way of the **serviceDeploy** or **createVersionedSCAModule** command) to ensure it is unique within the cell. This change is accomplished by adding the version number, a unique cell ID, or both to the original module name.

moduleName_vversionValue_uniqueCellID

Considerations for binding versioned modules

After you have deployed multiple versions of a module on a server or multiple instances of a module across clusters, consider how to bind specific versions of modules to clients (which might or might not be versioned).

- Static binding: If you are using static binding, use the existing administrative tools to bind a versioned module to a client. You must specify the module version number in the static binding.

- Dynamic binding: To use dynamic binding with versioned modules, use a mediation flow component that contains the module version metadata (versionValue and versionProvider) and service-version-aware routing. Note that in order to use service-version-aware routing to dynamically bind versioned modules, all modules must be registered with WebSphere Service Registry and Repository (WSRR).

Service application features of the administrative interfaces

Use the administrative console in WebSphere ESB to view and change aspects of service applications and service modules.

Service applications provide services, and have an associated service module, also called a Service Component Architecture (SCA) module.

The only type of SCA modules that are supported by WebSphere ESB are *mediation modules*.

Viewable module details

After you have deployed an enterprise archive (EAR) file containing an SCA module, you can view SCA module details. You can list all your SCA modules, and their associated applications, and you can view details about a particular SCA module.

You can view the following SCA module details:

- SCA module name
- Associated application
- SCA module imports:
  - Interfaces
  - Bindings
- SCA module exports:
  - Interfaces
  - Bindings
- SCA module properties

Modifiable module details

After you have deployed an EAR file containing an SCA module you can use the administrative console to change the following SCA module details. You do not need to redeploy the EAR file.

- Import bindings of type SCA:
  - Import bindings define service interactions. You can change the bindings if you want to change the service interactions.
  - SCA bindings connect SCA modules to other SCA modules. One SCA module can interact with a second SCA module, and can be changed to interact with another SCA module.
  - Web service bindings connect SCA modules to external services using SOAP.
- Import bindings of type Web service (WS):
  - Import bindings define service interactions. You can change the bindings if you want to change the service interactions.
  - SCA modules use WS import bindings to access web services. A WS import binding calls a service located at a specified endpoint. You can change the

end point so that the binding calls the service at an alternative end point, or calls an entirely different service with compatible interfaces.

- Export and import bindings of types JMS, WebSphere MQ JMS, generic JMS, WebSphere MQ, and HTTP have attributes that you can modify.
- Mediation module properties:
  - Mediation module properties belong to the mediation primitives with which they are associated. However, the administrative console displays some of them as Additional Properties of an SCA module. The integration developer must flag a mediation primitive property as Promoted in order for it to be visible.
  - Mediation module properties affect the behavior of your mediations. The mediation changes that you can make depend upon the properties that have been promoted.

Important: An export with no binding specified is interpreted by the run time as an export with an SCA binding.

Figure 134. Example showing one mediation module interacting with another mediation module. Mediation Module1 connects to Mediation Module2

Administering service modules in the administrative console

You can list the service modules that have been deployed to WebSphere ESB, view information associated with individual service modules, and make changes to some import bindings.

About this task

After deploying service applications, use the administrative console to list and administer the associated mediation modules.

Procedure

1. Open the administrative console.
2. Click **Applications** > **SCA Modules** to list the available service modules.

Results

The content pane displays the mediation modules that have been deployed to WebSphere Enterprise Service Bus. You can also see the applications that the mediation modules are associated with, and whether those applications are running.

Displaying details of a service module:

You can display information on service modules that have been deployed to WebSphere ESB.

About this task

To display details about the deployed service module use the administrative console to complete the following steps.

Procedure

1. In the navigation pane, expand **Applications** > **SCA Modules**, to display the SCA modules.
2. In the content pane, click the SCA module to choose an SCA module.

Results

The content pane displays the SCA module name and description; the name of the associated enterprise application; expandable lists of imports and exports; and a module properties link.

Displaying details of the application for a service module:

You can display details about the application used to deploy a service module to WebSphere ESB.

About this task

The application used to deploy a service module defines a range of configuration properties that affect the use of the module and associated components. When you installed the application, you specified most, if not all, of its property values.

After installing an application, you might want to review the properties and, if needed, change some of the values.

To display details about the application used to deploy a service module, use the administrative console to complete the following steps.

Procedure

1. Expand **Applications** > **SCA Modules** in the navigation pane to display the SCA modules.
2. In the column labeled **Application**, click the application name, to choose an SCA module.

Results

The content pane displays the application details page that provides the application's configuration properties and, if appropriate, local topology.

What to do next

From this page, you can review and, if needed, change the configuration properties for the application and link to additional console pages, as described in Configuring an application.

Starting and stopping service modules:

You can start a service module that has a status of Stopped or stop one that is running that has a status of Started. To change the status of a service module, start or stop the application used to deploy the module.

Before you begin

Before you can start or stop the application for a service module, you must have deployed the module into WebSphere ESB.

About this task

To use the services of a service module and associated components, start the associated application. By default, the application starts automatically when the server starts.

You can manually start and stop applications using the following administrative tools:
- Administrative console
- wsadmin startApplication and stopApplication commands
- Java programs that use ApplicationManager or AppManagement MBeans

To start or stop a service module, use the administrative console you complete the following steps.

Procedure

1. Expand **Applications** > **SCA Modules**, in the navigation pane, to list the SCA modules.
2. Select the check box for the SCA module that you want to start or stop.
3. Click the Start or Stop button

Option	Description
Start	Runs the application and changes the state of the application to *Started*. The status is changed to *partially started* if not all servers on which the application is deployed are running.
Stop	Stops the processing of the application and changes the state of the application to *Stopped*.

4. Click **Stop** or select the application you want to restart, then click **Start** to restart a running application.

Results

The status of the application changes and a message stating that the application started or stopped displays at the top the page.

What to do next

You can change whether or not an application starts automatically when the server on which it resides starts. For more information about starting and stopping WebSphere applications, see Starting and stopping applications.

Displaying service module properties:

You can display the properties of service modules that have been deployed to WebSphere ESB.

About this task

You might want to check that property values are what you expect before running a service application.

To display the properties of deployed service modules, use the administrative console to complete the following steps.

Procedure

1. In the navigation pane, expand **Applications** > **SCA Modules** to display the SCA modules
2. Click the required SCA module, in the content pane, to choose an SCA module.
3. Click **Module Properties**, under Additional Properties, in the content pane, to list the SCA module properties.
4. Optional: Expand the group whose properties you want to view. If properties belong to a group they are displayed inside an expandable section; if they do not belong to a group you can view them immediately.

Results

The content pane displays the updatable properties for the SCA module in a table that shows property names, types and values. Only property values are updatable from the administrative console: to change property groups, names, and types you use Integration Designer. A message is displayed if there are no properties that you can update.

Changing service module properties:

You can change the value of some service module properties.

About this task

You might want to change property values if the runtime environment changes.

To change the values of service module properties, use the administrative console to complete the following steps.

Procedure

1. Expand **Applications** > **SCA Modules** in the navigation pane to list the SCA modules.
2. Click an SCA module in the content pane to choose the SCA module.
3. Under Additional Properties, select **Module Properties**, in the content pane, to display the SCA module properties. This displays the module properties that you can update. Property groups, names, types, and values are displayed in the content pane, but you can only update property values. To change property groups, names, and types use Integration Designer.
4. Optional: Expand the group whose properties you want to update. If properties belong to a group they are displayed inside an expandable section; if they do not belong to a group you can view them immediately.
5. Click a property value from the Properties table to choose a property value.
6. Enter a value that conforms to the property type to change a property value.
7. Click **OK** to save your changes. Then save your changes to the master configuration.

Results

The property values are changed. Generally, mediation flows use property changes immediately, unless the changes occur in a deployment manager cell. If changes occur in a deployment manager cell, they take effect on each node in the cell after that node has been synchronized. Mediation flows that are in-flight at the time of the property value change continue to use previous values.

Working with imports and exports:

You can list the imports and exports of service modules that have been deployed to WebSphere ESB. You can also display import and export interfaces and change the details of import bindings and selected export bindings.

Displaying an import or export interface:

You can display the import or export interfaces of service modules that have been deployed to WebSphere ESB.

About this task

To display the import or export interfaces of service modules that you have deployed, use the administrative console to complete the following steps.

Procedure

1. From the console navigation pane, click **Applications** > **SCA Modules** > ***moduleName*** to display the SCA Modules detail page for that module.
2. From the SCA Modules detail page, do one of the following tasks, depending on the type of interface you want to view.

Option	Description
Viewing an import interface	1. In the content pane, expand **Imports** to list all imports associated with the module. 2. Expand the import you want to view, and then expand **Interfaces** to display the import interfaces. 3. Select the interface you want to display.
Viewing an export interface	1. In the content pane, expand **Exports** to list all exports associated with the module. 2. Expand the export you want to view, and then expand **Interfaces** to display the export interfaces. 3. Select the interface you want to display.

Results

The content pane displays the WSDL (Web Services Description Language) interface.

Displaying an import or export binding:

You can display details about import and export bindings after you deploy service modules to WebSphere ESB.

About this task

To display the import or export binding details of service modules that you have deployed, use the administrative console to complete the following steps.

Procedure

1. From the console navigation pane, click **Applications** > **SCA Modules** > ***moduleName*** to display the SCA Modules detail page for that module.
2. From the SCA Modules detail page, do one of the following tasks, depending on the type of binding you want to view.

Option	Description
Viewing an import binding	1. In the content pane, expand **Imports** to list all imports associated with the module. 2. Expand the import you want to view, and then expand **Bindings** to display the import bindings. 3. Select the binding you want to display.
Viewing an export binding	1. In the content pane, expand **Exports** to list all exports associated with the module. 2. Expand the export you want to view, and then expand **Bindings** to display the export bindings. 3. Select the binding you want to display.

Results

The content pane displays the import or export binding details.

Administering bindings:

You can display information about import and export bindings and, in some cases, you can update the properties of the bindings. You use the administrative console to display and change information related to bindings. You can also use commands to show and modify import and export binding information.

Administering SCA bindings:

You can view information about the SCA import and export bindings of a module after the module has been deployed to the server. You can also reconfigure selected properties of an SCA import binding.

Viewing and updating SCA import bindings:

Using the administrative console, you can view information about a Service Component Architecture (SCA) import binding and change the target of the associated module.

Before you begin

To perform this task, you must have permission to change the master configuration.

About this task

To view information about an SCA import binding or to change the target of the associated module, use the administrative console to complete the following steps.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports**.
   b. Expand the import, and then expand **Binding**.
   c. Click the binding to view information about its properties.
      - **Module** identifies the module that contains the import with this import binding.
      - **Version** displays the SCA module version, if the module is versioned.
      - **Cell ID** identifies the SCA module instance in the cell.
      - **Import** identifies the import that contains the selected import binding.
      - **Import interfaces** contains the list of interfaces for the import of this module.
3. To select a new target SCA module, perform the following steps:
   a. Select a module from the **Target** list.

      Selecting another SCA module changes the exports and export interfaces that are displayed.
   b. Select an export from the **Export** list.
4. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Viewing SCA export bindings:

Using the administrative console, you can view information about a Service Component Architecture (SCA) export binding, such as the name of the associated module and the name of the Web Services Description Language (WSDL) file.

About this task

To view information about an SCA export binding, use the administrative console to complete the following steps.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Exports**.
   b. Expand the export, and then expand **Binding**.
   c. Click the binding to view information about its properties.
      - **Module** identifies the module that contains the export with this export binding.
      - **Version** displays the SCA module version, if the module is versioned.
      - **Cell ID** identifies the SCA module instance in the cell.
      - **Export** identifies the export that contains the selected export binding.
      - **Export interfaces** contains the list of interfaces for the export of this module.

Administering Web service bindings:

You can view information about the Web service import and export bindings of a module after the module has been deployed to the server. You can also reconfigure selected properties of import bindings and configure policy sets for the bindings.

Viewing and updating Web service import bindings:

Using the administrative console, you can view information about a Web service import binding and change the endpoint URL. For the Java API for XML Web Services (JAX-WS) bindings, you can also configure a policy set for the binding.

Before you begin

To perform this task, you must have permission to change the master configuration.

About this task

The steps for administering Web service bindings depend on the type of binding:

- For JAX-RPC bindings, you can view attributes of the binding and edit the target endpoint.
- For JAX-WS bindings, you can view attributes of the binding, edit the target endpoint, and configure policy sets.

A *policy set* is a collection of policy types, each of which provides a quality of service. These types have been configured and can be associated with a Web service provider or consumer.

Policy sets work in pairs. You must have the same policy set on the service requester as on the service provider. Therefore, you should have the same policy set on the import binding as on the service provider it is calling.

Note: The **Policy set attachments** section of the administrative console page appears only for JAX-WS bindings. It does not appear for JAX-RPC service bindings.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports**.
   b. Expand the import, and then expand **Binding**.
   c. Click the binding to view information about its properties.
3. Change the value of **Target endpoint address**, which is the location of the Web service, and then click **Apply** or **OK**.
4. For JAX-WS bindings only, configure policy sets for import bindings by performing the following tasks:
   a. Optional: Expand **Preferences**, indicate the maximum number of rows and whether you want to retain the filter criteria, and click **Apply**.
   b. Optional: Select the filter icon if you want to use a filter to search the table.
   c. Select the import binding, and click **Attach** to attach a policy set to the binding, or click **Detach** to remove the policy set.
   d. To assign a policy set binding, select the import binding, click **Assign Binding**, and provide a name for the policy set binding.
   e. Repeat steps 4c and 4d for each binding you want to configure.
5. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Viewing and updating Web service export bindings:

Using the administrative console, you can view information about a Web service export binding (including the WSDL file) and configure properties of the associated Web module. For the Java API for XML Web Services (JAX-WS) bindings, you can also configure a policy set for the binding.

Before you begin

To perform this task, you must have permission to change the master configuration.

About this task

The steps for administering Web service bindings depend on the type of binding:

- For JAX-RPC bindings, you can view attributes of the binding.

- For JAX-WS bindings, you can view attributes of the binding and configure policy sets.

A *policy set* is a collection of policy types, each of which provides a quality of service. These types have been configured and can be associated with a Web service provider or consumer.

Policy sets work in pairs. You must have the same policy set on the service requester as on the service provider. Therefore, you should have the same policy set on the export binding as on the client.

Note: The **Policy set attachments** section of the administrative console page appears only for JAX-WS bindings. It does not appear for JAX-RPC service bindings.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Exports**.
   b. Expand the export, and then expand **Binding**.
   c. Click the binding to view information about its properties.
      - In the **General Properties** section, view the name, port, and location (endpoint address) of the Web service.
      - From the **Related Properties** list, click the interface to view the Web Services Description Language (WSDL) file that is associated with the Web service.
3. To change properties that are associated with the Web module, click one of the following properties in the **Web Module Properties** list:
   - Click **Manage Export Binding Web Module** to view or edit deployment-specific information for a Web module. For example, you can edit the **Starting weight**, which specifies the priority of this module during server startup.
   - Click **Context Root** to view the Web module name and Uniform Resource Identifier (URI) and edit the context root.
   - Click **Virtual Hosts** to specify the virtual host for the Web module. Virtual hosts let you associate a unique port with a module or application.
   - Click **JSP reload options for Web modules** to specify information about the reloading of JavaServer Pages (JSP) files (such as the number of seconds to scan a file system for updated JSP files).
   - Click **Session management** to specify information about HTTP session support. For example, you can set the number of minutes before a session times out.
4. For JAX-WS bindings only, configure policy sets for export bindings by performing the following tasks:
   a. Optional: Expand **Preferences**, indicate the maximum number of rows and whether you want to retain the filter criteria, and click **Apply**.
   b. Optional: Select the filter icon if you want to use a filter to search the table.
   c. Select the export binding, and click **Attach** to attach a policy set to the binding, or click **Detach** to remove the policy set.

d. To assign a policy set binding, select the export binding, click **Assign Binding**, and provide a name for the policy set binding.
e. Repeat steps 4c on page 422 and 4d for each binding you want to configure.
f. Save the changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Working with Web service (JAX-WS) bindings:

When you use Web service (JAX-WS) bindings with your applications, you can add a Security Assertion Markup Language (SAML) quality of service (QOS) to the binding. You must first use the administrative console to import the policy set. You can also use the administrative console to make sure that the server is properly configured for use with the Web service (JAX-WS) binding.

Importing SAML policy sets:

Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information. When you configure a Web service (JAX-WS) binding in Integration Designer, you can specify an SAML policy set. You first use the administrative console of WebSphere ESB to make the SAML policy sets available so that they can be imported into Integration Designer.

Before you begin

The SAML policy sets are typically located in the profile configuration directory:

profile_root/config/templates/PolicySets

Before you begin this procedure, verify that the following directories (which contain the policy sets) are located in the profile configuration directory:

- SAML11 Bearer WSHTTPS default
- SAML20 Bearer WSHTTPS default
- SAML11 Bearer WSSecurity default
- SAML20 Bearer WSSecurity default
- SAML11 HoK Public WSSecurity default
- SAML20 HoK Public WSSecurity default
- SAML11 HoK Symmetric WSSecurity default
- SAML20 HoK Symmetric WSSecurity default
- Username WSHTTPS default

If the directories are not in the profile configuration directory, copy them to that directory from the following location:

app_server_root/profileTemplates/default/documents/config/templates/PolicySets

About this task

You import the policy sets into the administrative console, select the ones you want to make available to Integration Designer, and then save a zip file for each of those policy sets to a location that is accessible by Integration Designer.

Procedure

1. Import the policy sets by following these steps:
   a. From the administrative console, click **Services** > **Policy Sets** > **Application policy sets**.
   b. Click **Import** > **From Default Repository**.
   c. Select the SAML default policy sets, and click **OK**.
2. Export the policy sets so that they can be used by Integration Designer:
   a. From the Application policy sets page, select the SAML policy set you want to export, and click **Export**.

      Note: If the Application policy sets page is not currently displayed, click **Services** > **Policy Sets** > **Application Policy Sets** from the administrative console.
   b. On the next page, click the .zip file link for the policy set.
   c. In the File Download window, click **Save** and indicate a location that is accessible by Integration Designer.
   d. Click **Back**.
   e. Complete steps 2a on page 82 through 2d on page 82 for each policy set you want to export.

Results

The SAML policy sets are saved in .zip files and are ready to be imported into Integration Designer.

What to do next

Import the policy sets into Integration Designer, as described in the topic “Policy sets”.

Invoking web services that require HTTP basic authentication:

HTTP basic authentication employs a user name and password to authenticate a service client to a secure endpoint. You can set up HTTP basic authentication when sending or receiving web service requests.

About this task

You set up HTTP basic authentication for receiving web service requests by configuring the Java API for XML Web Services (JAX-WS) export binding, as described in the “Creating and assigning security roles to web service exports” topic.

HTTP basic authentication can be enabled for web service requests that are sent by a JAX-WS import binding in one of two ways:

- When configuring the import binding in an SCA module, you can select the supplied HTTP authentication policy set named BPMHTTPBasicAuthentication (which is provided with the web service (JAX-WS) import binding) or any other policy set that includes the HTTPTransport policy.
- When constructing the SCA module, you can use mediation flow capabilities to dynamically create a new HTTP authentication header and specify the user name and password information in the header.

Note: The policy set has precedence over the value specified in the header. If you want to use the value set in the HTTP authentication header at run time, do not attach a policy set that includes the HTTPTransport policy. Specifically, do not use the default BPMHTTPBasicAuthentication policy set, and, if you have defined a policy set, make sure it excludes the HTTPTransport policy.

You can find more information about web service policy sets and policy bindings and how they are used in the "Web services policy sets" topic of the WebSphere Application Server Information Center. The link to the topic is listed under "Related information".

Procedure

- To use the supplied policy set, perform the following steps:
  1. Optional: In the administrative console, create a client general policy binding or edit an existing one that includes the HTTPTransport policy with the desired user ID and password values.
  2. In IBM Integration Designer, generate a web service (JAX-WS) import binding and attach the BPMHTTPBasicAuthentication policy set.
  3. Perform *one* of the following steps:
     - In IBM Integration Designer, in the web service (JAX-WS) import binding properties, specify the name of an existing client general policy binding that includes the HTTPTransport policy.
     - After deploying the SCA module, from the administrative console of the process server, select an existing client policy binding or create a new one and associate it with the import binding.
  4. Optional: In the administrative console of the process server, edit the selected policy set binding to specify the required ID and password.
- To specify the user name and password in the HTTP authentication header, perform one of the following sets of steps:
  - Use the HTTP Header Setter mediation primitive in IBM Integration Designer to create the HTTP authentication header, and specify the user name and password.
  - If additional logic is required, use Java code in a custom mediation primitive (as shown in the following example) to:
    1. Create an HTTP authentication header.
    2. Specify the user name and password information.
    3. Add the new HTTP authentication header to HTTPControl.
    4. Set the updated HTTPControl back in the Context service.

```
//Get the HeaderInfoType from contextService
ContextService contextService = (ContextService) ServiceManager.INSTANCE
.locateService("com/ibm/bpm/context/ContextService");
HeaderInfoType headers = contextService.getHeaderInfo();
if(headers == null){
    headers = ContextObjectFactory.eINSTANCE.createHeaderInfoType();
}
```

```
//Get the HTTP header and HTTP Control from HeaderInfoType
HTTPHeaderType httpHeaderType = headers.getHTTPHeader();
HTTPControl cp = httpHeaderType.getControl();
HeadersFactory factory = HeadersFactory.eINSTANCE;
if(cp == null){
    cp = factory.createHTTPControl();
}
//Create new HTTPAuthentication and set the HTTPCredentials
HTTPAuthentication authorization = factory.createHTTPAuthentication();
HTTPCredentials credentials = factory.createHTTPCredentials();
authorization.setAuthenticationType(HTTPAuthenticationType.BASIC_LITERAL);
credentials.setUserId("USERNAME");
credentials.setPassword("PASSWORD");
authorization.setCredentials(credentials);
cp.setAuthentication(authorization);
httpHeaderType.setControl(cp);
// Set header info back to the current execution context.
contextService.setHeaderInfo(headers);
```

Checking the server configuration:

When you deploy an application with a Web service (JAX-WS) binding, you must make sure that the server on which the application is deployed does not have the **Start components as needed** option selected.

About this task

You can check to see whether this option is selected by performing the following steps from the administrative console:

Procedure

1. Click **Servers** > **Server types** > **WebSphere application servers**.
2. Click the server name.
3. From the Configuration tab, determine whether **Start components as needed** is selected.
4. Perform one of the following steps:
   - If **Start components as needed** is selected, remove the check, and then click **Apply**.
   - If **Start components as needed** is not selected, click **Cancel**.

Administering HTTP bindings:

You can view information about the HTTP import and export bindings of a module after the module has been deployed to the server. You can also tune, or set, special features of the import and export bindings.

You use Integration Designer to create HTTP imports and exports.

Viewing and updating HTTP import bindings:

Using the administrative console, you can change the configuration of HTTP import bindings without changing the original source and then redeploying the application.

About this task

Change HTTP import bindings when the binding properties of an HTTP application used by a Service Component Architecture (SCA) module change.

Required security role for this task: When security and role-base authorization are enabled, you must log in as an administrator or configurator to perform this task.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports**.
   b. Expand the import, and then expand **Binding**.
   c. Click the binding to view information about its properties.
3. Select the scope for the changes you want to make.
   - To change the configuration on the binding scope, click the **Binding Scope** tab.
   - To change the configuration at the method scope, click the **Method Scope** tab.

   When both configurations exist, the method scope configuration takes precedence over the binding scope configuration.
4. Make changes to one or more of the following properties:
   - **Select method** (Method scope only)

     Choose the method that you want to review or configure. Click on the arrow in the **Select method** field to see the list of methods that can be configured.
   - **Endpoint URL**

     Specifies the URL for the target service.
   - **HTTP Method**

     Specifies the method to use for the endpoint URL.
   - **HTTP version**

     Specifies the HTTP version to use for this endpoint URL. The choices are **1.0** and **1.1**. The default is **1.1**.
   - **Number of connection retries**

     Specifies the number of times the request is retried when the system receives an error response. The default is **0**, which means that, after a failure, no attempt is made.
   - **Basic HTTP authentication**

     Specifies the authentication alias to use with the HTTP server on this binding.

     To choose the authentication alias, select the alias name from the list. To change the attributes of a selected authentication alias, click **Edit**. To create a new authentication alias, click **New**.
   - **SSL authentication**

     Specifies the Secure Sockets Layer (SSL) configuration to use for this binding.

     To edit an existing configuration, select the name from the list and click **Edit**. To create a new configuration, click **New**.
   - **Transfer Encoding**

     Specifies how information is transferred between the endpoints. Choices are **chunked** or **identity**.

     The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size. This allows dynamically

produced content to be transferred along with the information necessary for the recipient to verify that it has received the full message.

Important: If you set this parameter to **chunked**, Content Encoding is set to **identity** and you will be unable to change Content Encoding.

- **Content Encoding**

  Specifies how the content that traverses the binding is encoded. Choose either **gzip**, **x-gzip**, **deflate**, or **identity**.

- **HTTP proxy settings** or **HTTPS proxy settings**

  Specifies the settings for bindings that do not require security authorization for access (**HTTP proxy settings**) or that do require authorization for access (**HTTPS proxy settings**).

  – **Proxy host**

    Specifies the host name or IP address of an HTTP proxy server through which to connect to the endpoint URL.

  – **Proxy port**

    Specifies the port used to connect to an HTTP proxy server for this binding.

  – **Proxy credentials**

    Specifies the Java2 Connectivity (J2C) authentication alias to use for the proxy settings.

    To change an existing alias, select the alias from the list and click **Edit**. To add a new alias, click **New**.

  – **Non-proxied hosts**

    Specifies a list of hosts on this binding that do not use proxies. Enter each host on a separate line (use the Enter key).

    To add a host to the list, type the host at the end of the list, separating it from the previous entry by clicking the Enter key. To remove a host from the list, delete the host from the list.

- **Response read timeout**

  Specifies the time, in seconds, that the binding waits to read the response message from the HTTP transport channel. Setting this field to **0** causes the binding to wait indefinitely.

  Note: The HTTP transport channel has its own Read timeout value, which is the amount of time that the channel waits for a read request to complete on a socket after the first read request occurs. This setting is described in HTTP transport channel settings in the WebSphere Application Server Information Center.

5. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Viewing and updating HTTP export bindings:

Using the administrative console, you can change the configuration of HTTP export bindings without changing the original source and then redeploying the application.

About this task

Change HTTP export bindings when you need to change whether a method on a binding is pingable or to change the encodings a method or binding supports.

Required security role for this task: When security and role-base authorization are enabled, you must log in as an administrator or configurator to perform this task.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Exports**.
   b. Expand the export, and then expand **Binding**.
   c. Click the binding to view information about its properties.
3. Select the scope for the changes you want to make.
   - To change the configuration on the binding scope, click the **Binding Scope** tab.
   - To change the configuration at the method scope, click the **Method Scope** tab.

   When both configurations exist, the method scope configuration takes precedence over the binding scope configuration.
4. Make changes to one or more of the following properties:
   - **Select method** (Method scope only)

     Choose the method that you want to review or configure. Click on the arrow in the **Select method** field to see the list of methods that can be configured.
   - **HTTP Methods**

     Lists the methods and the current configuration for the methods. You can set whether the method is pingable and the return code for the method.
     – **Method**

       The name of the method. The methods are GET, POST, PUT, DELETE, TRACE, OPTIONS, and HEAD.
     – **Pingable**

       Whether or not an HTTP client can ping the method. When selected, you must specify the return code the binding returns to the client. The default for this is unchecked.
     – **Return code**

       An integer returned when an HTTP client pings the method.
   - **Transfer Encoding**

     Specifies how information is transferred between the endpoints. Choices are **chunked** or **identity**.

     The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size. This allows dynamically

produced content to be transferred along with the information necessary for the recipient to verify that it has received the full message.

Important: If you set this parameter to **chunked**, Content Encoding is set to **identity** and you will be unable to change Content Encoding.

- **Content Encoding**

  Specifies how the content that traverses the binding is encoded. Choose either **gzip**, **x-gzip**, **deflate**, or **identity**.

5. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Administering EJB bindings:

You can view information about the EJB import and export bindings of a module after the module has been deployed to the server. You can also reconfigure selected properties of import bindings.

Viewing and updating EJB import bindings:

You can view information about EJB import bindings using the WebSphere administrative console. You can also modify the JNDI name associated with the binding.

Before you begin

To see or edit an EJB binding, it must be installed as part of a Service Component Architecture (SCA) application in your server profile.

About this task

You can modify only the **JNDI name** property. All other properties for an EJB binding import are read-only.

EJBs invoked by an EJB import can be running in any of the following combinations. For each one of these scenarios, make sure you consider the information in the JNDI configuration information column when you modify the JNDI name:

Table 26. EJB import JNDI name configurations

EJB scenario	JNDI configuration information
WebSphere ESB in a different Java EE module	Set the JNDI name in the EJB import binding to match the global namespace. Also, confirm that the JNDI name specified in the EJB import binding matches what is specified in the Java EE module bindings file. **Note:** The JNDI name for local invocations, which apply only to the EJB 3.0 programming model, take the form `ejblocal:` followed by the fully qualified class name of the local interface. You can find more information in the "JNDI name" topic.
Remote WebSphere ESB or WebSphere Application Server	Create a namespace binding (of EJB binding type) using the WebSphere ESB administrative console. To create the namespace binding, click **Environment** > **Naming** > **Namespace**. The name specified in the **namespace** field for the namespace binding should match the JNDI name specified in the EJB import binding configuration.
Remote Java EE server (other than WebSphere ESB or WebSphere Application Server)	Create a namespace binding using the WebSphere ESB administrative console. • If the Java EE server provides a COSNaming interface, create a namespace binding of type CORBA. • If the Java EE server does not provide a COSNaming interface, create a namespace binding of the indirect type. To create the namespace binding, click **Environment** > **Naming** > **Namespace**. The name specified in the **namespace** field for the namespace binding should match the JNDI name specified in the EJB import binding configuration.

If your implementation involves WebSphere Application Server, additional configuration using the WebSphere Application Server administrative console might be required.

To view or configure EJB import properties using the WebSphere ESB administrative console, complete the following steps:

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports**.
   b. Expand the import, and then expand **Binding**.
   c. Click the binding to view information about its properties.
3. Optional: Change the JNDI name.
4. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Viewing EJB export bindings:

You can view EJB export bindings using the WebSphere administrative console.

Before you begin

To see an EJB export binding, it must be installed as part of a Service Component Architecture (SCA) application in your server profile.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Exports**.
   b. Expand the export, and then expand **Binding**.
   c. Click the binding to view information about its properties.

Administering EIS bindings:

EIS bindings are installed in the server as part of your SCA applications. Administer your bindings from the administrative console.

Before you begin

You must have permission to make changes to the master configuration in order to perform this task.

About this task

You have an installed application that includes an EIS import or export module.

To change configuration properties after you deploy the adapter as part of a module, you use the administrative console of the runtime environment. You can update resource adapter properties (used for general adapter operation), managed connection factory properties (used for outbound processing), and activation specification properties (used for inbound processing).

Note: You can also set configuration properties after you install a stand-alone adapter. To do so, from the administrative console, expand **Resources** > **Resource adapters**, and select the adapter whose properties you want to configure.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports** or **Exports**.
   b. Expand the import or export, and then expand **Binding**.
   c. To view the WSDL, expand **Interfaces** and select the interface you want to view. The WSDL of the interface is displayed. The WSDL cannot be edited through the administrative console but can be altered with text editors.
   d. To view the binding, expand **Bindings** and click the import or export binding that you want to view. You can change the port or the name of the imported or exported service.
3. Optional: Change the port or the name of the imported or exported service.
4. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Administering JMS bindings:

You can view information about the JMS import and export bindings of a module after the module has been deployed to the server. You can also tune, or set, special features of the import and export bindings.

Use the administrative console to configure and administer JMS import and export bindings.

For detailed instructions on generating JMS imports and exports, see “Generating a JMS import binding” and “Generating a JMS export binding” in the Integration Designer information center.

Viewing and updating JMS bindings:

You can configure JMS import and export bindings to apply special features of the resource. The administrative tasks are performed using the WebSphere administrative console.

Before you begin

You must have permission to make and save changes to the profile on the administrative console.

About this task

The JMS import or export must be installed as part of a Service Component Architecture (SCA) application in your server profile.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports** or **Exports**.
   b. Expand the import or export, and then expand **Binding**.
   c. Click the binding to view information about its properties. The general properties of the binding are displayed:
      - The **Send Resources** category contains the Connection Factory and the Send Destination.
      - The **Receive Resources** category contains the Response Connection Factory and the Activation Specification.
      - The **Advanced Resources** category contains Callback resources and other available resources.

      Note: You can also access a resource by typing the JNDI name in the text box. Doing so, however, allows you to enter the name of a resource that is not yet configured.
3. Make any desired changes to the resources:
   a. Click **Browse** to open a window with a list of JNDI names; then, select the desired JNDI name.
   b. Click **Configure** to open the corresponding page referred to by the JNDI name.
4. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Properties of JMS bindings:

The JMS import and export bindings can be installed with all the necessary connection factories having been created during deployment, or they can be configured to point to a set of existing resources.

Typically, JMS import and export bindings are created in Integration Designer. When you configure the binding, you can either create the connections and destinations required for the JMS binding (by selecting **Configure new messaging provider resources**, which is the default), or you can select **Use pre-configured messaging provider resources**. If you choose pre-configured, you add the JNDI

names for the connection factory and the send destination (for a one-way operation) or the send and receive destinations (for a request-response operation).

Configuring the JMS binding depends upon which option was selected.

Table 27 and Table 28 show examples of the resources you specify when you select **Use pre-configured messaging provider resources**.

Note: The JNDI name takes the form:

`moduleName/importName_resourceAbbreviation`

or

`moduleName/exportName_resourceAbbreviation`

For example, for a module named `Inventory` and an import named `Import1`, the JNDI name for the connection factory would be:

`Inventory/Import1_CF`

The fields and associated values for import bindings are shown in the following table:

Table 27. Example values for import bindings

Property	Example
JNDI name for connection factory	*moduleName/importName*_CF
JNDI name for send destination	*moduleName/importName*_SEND_D
JNDI name for receive destination	*moduleName/importName*_RECEIVE_D

The fields and associated values for export bindings are shown in the following table:

Table 28. Example values for export bindings

Property	Example
JNDI name for activation specification	*moduleName/exportName*_AS
JNDI name for receive destination	*moduleName/exportName*_RECEIVE_D
JNDI name for send destination	*moduleName/exportName*_SEND_D

Note: The resources are created at the server scope. The scope in the administrative console is initially set to **All Scopes**. You must set the scope to **cell** or **node** to create a new resource. You can select an existing resource from the default list.

Viewing or changing the state of an endpoint:

You can use the administrative console to manage the state of endpoints associated with JMS bindings, WebSphere MQ JMS bindings, and WebSphere MQ bindings. For example, you can pause or resume an endpoint associated with one of those bindings.

About this task

The import or export must be installed as part of a Service Component Architecture (SCA) application in your server profile.

Note: This procedure applies only to Version 7 (or later) applications deployed to the runtime environment.

Procedure

1. Select the SCA module. From the administrative console, click **Applications** > **SCA Modules** and then click the *modulename*.
2. Under **Module components**, expand **Imports** or **Exports**.
3. Expand the import or export and then expand **Binding**. Make sure that you select one of the following bindings:
   - JMS
   - WebSphere MQ JMS
   - WebSphere MQ
4. Click on the binding to be administered.
5. To view or change the state of a binding endpoint, perform the following tasks:
   a. Click the **Runtime** tab.
   b. In the **Receiving Endpoints** table, select the check box for the endpoint.
   c. Click **Pause** or **Resume** to temporarily stop or restart the endpoint.

Results

The endpoint is paused or resumed.

Administering Generic JMS bindings:

You can view information about the Generic JMS import and export bindings of a module after the module has been deployed to the server. You can also tune, or set, special features of the import and export bindings.

Use the administrative console to configure and administer Generic JMS import and export bindings.

For detailed instructions on generating Generic JMS imports and exports, see "Generating a generic JMS import binding" and "Generating a generic JMS export binding" in the Integration Designer information center.

Setting up connectivity for the Generic JMS binding:

You must set up connectivity to and from a third-party JMS provider to use the Generic JMS binding.

Before you begin

You must have permission to make and save changes to the profile on the administrative console. You must have the appropriate permissions to make and save changes in IBM Integration Designer and in WebSphere Application Server.

About this task

This task provides a procedural outline only; providing specific instructions for individual third-party JMS providers is beyond the scope of this topic.

The application in this scenario contains a mediation component connection to other applications at both ends by means of the Generic JMS binding; the

application contains an interface with a single two-way operation.

Procedure

1. Configure your third-party JMS provider to create a queue manager, queues, and JMS connection factories and destinations using the provider-specific tooling.
2. In WebSphere Application Server, you must define a generic messaging provider.
3. In Integration Designer, you must perform the following tasks:
   a. Add an import and export to the application and connect them to a previously-implemented mediation component.
   b. Add a Generic JMS binding to both the export and the import: **Generate binding** > **Messaging binding** > **Generate JMS binding** .
   c. Set the genericMessagingProviderName property on both the import and export to match the properties previously defined to WebSphere Application Server.
   d. Set the ExternalJNDIName for the connections and send/receive destinations to match those defined in your third-party JMS provider tooling.
4. Deploy the application to a single server.

   Make sure that the third-party JMS provider queue manager is running and available for connection and that the context to which the generic messaging provider definition points in WebSphere Application Server is available.

   You can build and deploy your application using Integration Designer. Another way to deploy applications is to export the modules as zip files and then use the serviceDeploy command of IBM Process Server or WebSphere Enterprise Service Bus (mediation modules only) to build and deploy them as EAR files.
5. Start the application.
6. Run the application.

Results

The application can be run by placing messages on the third-party JMS provider queue defined in the Generic JMS export receive destination. Responses will be returned to the Generic JMS export send destination.

Similarly, the application will issue requests to the Generic JMS import send destination and expect responses on the Generic JMS import receive destination.

Viewing and updating Generic JMS bindings:

You can administer Generic JMS import and export bindings to configure special features of the resource. The administrative tasks are performed using the administrative console.

Before you begin

You must have permission to make and save changes to the profile on the administrative console, and you must have completed the connectivity setup procedure.

About this task

The Generic JMS import or export must be installed as part of a Service Component Architecture (SCA) application in your server profile.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports** or **Exports**.
   b. Expand the import or export, and then expand **Binding**.
   c. Click on the binding to be administered. The general properties of the binding are displayed:
      - The **Send Resources** category contains the Connection Factory and the Send Destination.
      - The **Receive Resources** category contains the Response Connection Factory, the Listener Port, and the Activation Specification.
      - The **Advanced Resources** category contains Callback resources and other available resources.

      Note: You can also access a resource by typing the JNDI name in the text box. Doing so, however, allows you to enter the name of a resource that is not yet configured.
3. Administer the desired resource:
   a. Click **Browse** to open a window with a list of JNDI names; then, select the desired JNDI name. The selected name will populate the appropriate text field.
   b. Click **Configure** to open the corresponding page referred to by the JNDI name. While most resources can be configured at cluster scope, selecting the **Configure** option at Listener Port displays a page showing all listener ports with the cluster Listener Port names for all the members of the given cluster; you can then select one listener port.

      When **Configure** has been selected, the corresponding WebSphere Application Server page will open.
4. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Properties of Generic JMS bindings:

The Generic JMS import and export bindings can be installed with all the necessary connection factories having been created during deployment, or they can be configured to point to a set of existing resources.

Typically, the Generic JMS bindings are created in Integration Designer. You can either create the connections and destinations required for the JMS binding at the time the component is installed on your server, or you can specify the JNDI name of the resources on the server that you intend your JMS import or export to use.

Configuring the Generic JMS binding depends upon which option was selected.

In the case where new message provider resources are created (that is, the resources are created on the server during installation), the resources will exist and can be located and administered using the administrative console. The JNDI names of the generated artifacts are described in the following tables.

Table 29. Generic JMS imports: Names and JNDI names of resources created at installation on the server

Resource	Generated resource JNDI name
Outbound Connection	[moduleName]/[importName]_CF
Response Connection	[moduleName]/[importName]_RESP_CF
Send destination	[moduleName]/[importName]_SEND_D
Receive destination	[moduleName]/[importName]_RECEIVE_D
Callback destination	[moduleName]/[importName]_CALLBACK_D

Table 30. Generic JMS exports: Names and JNDI names of resources created at installation on the server

Resource	Generated resource JNDI name
Inbound Connection	[moduleName]/[exportName]_LIS_CF
Response Connection	[moduleName]/[exportName]_RESP_CF
Receive destination	[moduleName]/[exportName]_RECEIVE_D
Send destination	[moduleName]/[exportName]_SEND_D
Callback destination	[moduleName]/[exportName]_CALLBACK_D

Note: The resources are created at the server scope. The scope in the administrative console is initially set to **All scopes**. You must set the scope to **cell** or **node** to create a new resource. You can select an existing resource from the default list.

If you select the other option and the JMS import is expecting to find required resources on the server, you must have these resources installed and the import and export files must contain the JNDI names. The association between the JMS binding and the resources will then be made.

Administering WebSphere MQ JMS bindings:

You can view information about the WebSphere MQ JMS import and export bindings of a module after the module has been deployed to the server. You can also tune, or set, special features of the import and export bindings.

Use the administrative console to access WebSphere MQ JMS bindings.

For detailed instructions on generating WebSphere MQ imports and exports, see “Generating an MQ JMS import binding” and “Generating an MQ JMS export binding” in the Integration Designer information center.

Viewing and updating MQ JMS bindings:

You can administer MQ JMS bindings to configure special features of the resource. The administrative tasks are performed using the administrative console.

Before you begin

You must have permission to make and save changes to the profile on the administrative console.

The queue and queue manager are not automatically generated; they must be created in WebSphere MQ by your WebSphere MQ administrator.

About this task

The MQ JMS import or export must be installed as part of a Service Component Architecture (SCA) application in your server profile.

Procedure

1. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
2. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports** or **Exports**.
   b. Expand the import or export, and then expand **Binding**.
   c. Click the binding to view information about its properties. The general properties of the binding are displayed:
      - The **Send Resources** category contains the Connection Factory and the Send Destination.
      - The **Receive Resources** category contains the Response Connection Factory and the Activation Specification.
      - The **Advanced Resources** category contains Callback resources and other available resources.

      Note: You can also access a resource by typing the JNDI name in the text box. Doing so, however, allows you to enter the name of a resource that is not yet configured.
3. Make any desired changes to the resources:
   a. Click **Browse** to open a window with a list of JNDI names; then, select the desired JNDI name. The selected name will populate the appropriate text field.
   b. Click **Configure** to open the corresponding page referred to by the JNDI name.

      Note: Most resources can be configured at the cluster scope. However, when you select the **Configure** option for the activation specification, a page is displayed that shows all activation specifications for all members of the given cluster; you can then select one activation specification.
4. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Properties of MQ JMS bindings:

MQ JMS bindings can be installed with all the necessary connection factories having been created during deployment, or they can be configured to point to a set of existing resources.

Typically, MQ JMS bindings are created in Integration Designer. You can either create the connections and destinations required for the JMS binding at the time the component is installed on your server, or you can specify the JNDI name of the resources on the server that you intend your MQ JMS import or export to use.

Configuring the MQ JMS binding depends upon which option was selected.

In the case where new message provider resources are created (that is, the resources are created on the server during installation), the resources will exist and can be located and administered using the administrative console.

Examples of the JNDI names of the generated artifacts are described in the following tables.

Table 31. MQ JMS imports: Names and JNDI names of resources created at installation on the server

Resource	Module name	Import name	Resource global JNDI name
Outbound Connection Factory	mqjms.module	my/import	mqjms.module/my/import_MQ_CF
Response Activation Specification	mqjms.module	my/import	mqjms.module/my/import_AS
Failed Event Replay Connection Factory	mqjms.module	my/import	mqjms.module/my/import_RESP_CF
Send	mqjms.module	my/import	mqjms.module/my/import_MQ_SEND_D
Receive	mqjms.module	my/import	mqjms.module/my/export_MQ_RECEIVE_D
SIB Callback Destination	mqjms.module	my/import	mqjms.module/my/import_MQ_CALLBACK_D
SIB Callback Connection Factory	All modules	my/import	SCA.MQJMS/Callback_CF

Table 32. MQ JMS exports: Names and JNDI names of resources created at installation on the server

Resource	Module name	Export name	Resource global JNDI name
Request Activation Specification	mqjms.module	my/export	mqjms.module/my/export_AS

Table 32. MQ JMS exports: Names and JNDI names of resources created at installation on the server (continued)

Resource	Module name	Export name	Resource global JNDI name
Failed Event Replay Connection Factory	mqjms.module	my/export	mqjms.module/my/export_LIS_CF
Response Connection Factory	mqjms.module	my/export	mqjms.module/my/export_RESP_CF
Receive	mqjms.module	my/export	mqjms.module/my/export_MQ_RECEIVE_D
Send	mqjms.module	my/export	mqjms.module/my/export_MQ_SEND_D
SIB Callback Destination	mqjms.module	my/export	mqjms.module/my/export_MQ_CALLBACK_D
SIB Callback Connection Factory	All modules	my/export	SCA.MQJMS/Callback_CF

Note:

- The resources are created at the server scope. The default scope in the administrative console is cell. You must change the scope in order to locate and administer the resources.
- The SIB callback destination and SIB callback connection factory are SIB JMS resources. The other entries in the table are MQ JMS resources. The two types of resources are administered separately in the administrative console.

If you select the other option and the MQ JMS import or export binding is expecting to find on the server resources that it will use, you must have these resources installed and the import file must contain their JNDI names. The association between the MQ JMS import and the resources will then be made.

Viewing or changing the state of an endpoint:

You can use the administrative console to manage the state of endpoints associated with JMS bindings, WebSphere MQ JMS bindings, and WebSphere MQ bindings. For example, you can pause or resume an endpoint associated with one of those bindings.

About this task

The import or export must be installed as part of a Service Component Architecture (SCA) application in your server profile.

Note: This procedure applies only to Version 7 (or later) applications deployed to the runtime environment.

Procedure

1. Select the SCA module. From the administrative console, click **Applications** > **SCA Modules** and then click the *modulename*.
2. Under **Module components**, expand **Imports** or **Exports**.
3. Expand the import or export and then expand **Binding**. Make sure that you select one of the following bindings:
   - JMS

- WebSphere MQ JMS
- WebSphere MQ

4. Click on the binding to be administered.
5. To view or change the state of a binding endpoint, perform the following tasks:
   a. Click the **Runtime** tab.
   b. In the **Receiving Endpoints** table, select the check box for the endpoint.
   c. Click **Pause** or **Resume** to temporarily stop or restart the endpoint.

Results

The endpoint is paused or resumed.

Administering WebSphere MQ bindings:

You can view information about the WebSphere MQ import and export bindings of a module after the module has been deployed to the server. You can also tune, or set, special features of the import and export bindings.

Use the administrative console to access WebSphere MQ bindings.

For detailed instructions on generating WebSphere MQ imports and exports, see "Generating an MQ JMS import binding" and "Generating an MQ JMS export binding" in the Integration Designer information center.

Viewing and updating WebSphere MQ bindings:

You can administer WebSphere MQ import and export bindings to tune, or set, special features of the resource. The administrative tasks are performed using the administrative console.

Before you begin

You must have permission to make and save changes to the profile on the administrative console.

The queue and queue manager are not automatically generated and must be created in WebSphere MQ by your WebSphere MQ administrator.

About this task

The WebSphere MQ import or export must be installed as part of a Service Component Architecture (SCA) application in your server profile.

Procedure

1. Open the default messaging provider settings page in the administrative console.

   Expand **JMS Providers** and click **WebSphere MQ**.
2. Optional: Administer WebSphere MQ connection factories.

   Click **WebSphere MQ connection factory** in the list of additional properties. This page shows a list of WebSphere MQ connection factories with a summary of their configuration properties. Click the MQ connection factory that you want to administer, or click **New** to create a new connection factory.

Use the subsequent page to browse or change the configuration properties of the selected connection factory for use with WebSphere MQ as a JMS provider. These configuration properties control how connections are created to associated queues.

You set these properties in the bindings for the resource reference of the application. If you do not want to modify the bindings for an existing application, locate this connection factory in the JCA pages where you can find these properties.

3. Optional: Administer WebSphere MQ queue connection factories.

   Click **WebSphere MQ queue connection factories** in the list of additional properties. This page shows a list of WebSphere MQ queue connection factories with a summary of their configuration properties. Click the WebSphere MQ queue connection factory that you want to administer, or click **New** to create a new queue connection factory.

   Use the subsequent page to browse or change the configuration of the selected queue connection factory for use with the WebSphere MQ JMS provider. These configuration properties control how connections are created to associated queues.

   A WebSphere MQ queue connection factory is used to create JMS connections to queues provided by WebSphere MQ for point-to-point messaging. Use WebSphere MQ queue connection factory administrative objects to manage queue connection factories for the WebSphere MQ JMS provider.

4. Optional: Administer WebSphere MQ queue destinations.

   Click **WebSphere MQ queue destinations** in the list of additional properties. This page shows a list of the WebSphere MQ queue destinations with a summary of their configuration properties. Click the queue destination that you want to administer, or click **New** to create a new WebSphere MQ queue destination.

   New WebSphere MQ queue destinations must be configured with the custom properties in the following table.

Table 33. Custom properties for WebSphere MQ queue destinations

Destination Type	Property Name	Property Value	Property Type
Send destination	MDWRITE	YES	java.lang.String
	MSGBODY	MQ	java.lang.String
	MDMSGCTX	SET_ALL_CONTEXT	java.lang.String
Receive destination	MDREAD	YES	java.lang.String
	MSGBODY	MQ	java.lang.String
	MDMSGCTX	SET_ALL_CONTEXT	java.lang.String

   Use the subsequent page to browse or change the configuration properties of the selected queue destination for point-to-point messaging with WebSphere MQ as a messaging provider.

   A WebSphere MQ queue destination is used to configure the properties of a queue. Connections to the queue are created by the associated queue connection factory for WebSphere MQ as a messaging provider.

5. Select the module that contains the binding by navigating to **Applications** > **SCA Modules** and clicking the module name.
6. Select the binding by performing the following steps:
   a. In the **Module components** section, expand **Imports** or **Exports**.
   b. Expand the import or export, and then expand **Binding**.

c. Click the binding to view information about its properties. The general properties of the binding are displayed:
   - The **Send Resources** category contains the Connection Factory and the Send Destination.
   - The **Receive Resources** category contains the Response Connection Factory and the Activation Specification.
   - The **Advanced Resources** category contains Callback resources and other available resources.

   Note: You can also access a resource by typing the JNDI name in the text box. Doing so, however, allows you to enter the name of a resource that is not yet configured.

7. Make any desired changes to the resources:
   a. Click **Browse** to open a window with a list of JNDI names; then, select the desired JNDI name. The selected name will populate the appropriate text field.
   b. Click **Configure** to open the corresponding page referred to by the JNDI name.

      Note: Most resources can be configured at the cluster scope. However, when you select the **Configure** option for the activation specification, a page is displayed that shows all activation specifications for all members of the given cluster; you can then select one activation specification.
8. Save your changes to the master configuration.

Results

If you made any updates, the binding is changed for the selected module. The changes take effect after you restart the SCA module.

Restriction: If the module is redeployed, the configuration changes you made are replaced by the original settings.

To ensure that the changes you made remain with the module across deployments, use IBM Integration Designer to make the changes in the source code for the module.

Properties of WebSphere MQ bindings:

The WebSphere MQ binding can be installed with all the necessary connection factories having been created during deployment, or they can be configured to point to a set of existing resources.

Typically, WebSphere MQ import and export bindings are created in Integration Designer. You can either create the connections and destinations required for the WebSphere MQ binding at the time the component is installed on your server, or you can specify the JNDI name of the resources on the server that you intend your WebSphere MQ binding to use.

Configuring the WebSphere MQ binding depends upon which option was selected.

In the case where new message provider resources are created (that is, the resources are created on the server during installation), the resources will exist and can be located and administered using the administrative console.

Examples of the JNDI names of the generated artifacts are described in the following tables.

Table 34. WebSphere MQ import: Names and JNDI names of resources created at installation on the server

Resource	Module name	Import name	Resource global JNDI name
Outbound Connection Factory	mq.module	my/import	mq.module/my/import_MQ_CF
Response Activation Specification	mq.module	my/import	mq.module/my/import_AS
Send	mq.module	my/import	mq.module/my/import_MQ_SEND_D
Receive	mq.module	my/import	mq.module/my/export_MQ_RECEIVE_D
SIB Callback Destination	mq.module	my/import	mq.module/my/import_MQ_CALLBACK_D
SIB Callback Connection Factory	All modules	my/import	SCA.MQ/Callback_CF

Table 35. WebSphere MQ export: Names and JNDI names of resources created at installation on the server

Resource	Module name	Export name	Resource global JNDI name
Request Activation Specification	mq.module	my/export	mq.module/my/export_AS
Response Connection Factory	mq.module	my/export	mq.module/my/export_RESP_CF
Receive	mq.module	my/export	mq.module/my/export_MQ_RECEIVE_D
Send	mq.module	my/export	mq.module/my/export_MQ_SEND_D
SIB Callback Destination	mq.module	my/export	mq.module/my/export_MQ_CALLBACK_D
SIB Callback Connection Factory	All modules	my/export	SCA.MQ/Callback_CF

Note:

- The resources are created at the server scope. The default scope in the administrative console is cell. You must change the scope in order to locate and administer the resources.
- The SIB Callback Destination and SIB Callback Connection Factory are SIB JMS resources. The other entries in the table are WebSphere MQ resources. The two types of resources are administered separately from the administrative console.

If you select the other option and the WebSphere MQ binding is expecting to find resources on the server that it will use, you must have these resources installed and the import or export file must contain their JNDI names. The association between the WebSphere MQ binding and the resources will then be made.

Viewing or changing the state of an endpoint:

You can use the administrative console to manage the state of endpoints associated with JMS bindings, WebSphere MQ JMS bindings, and WebSphere MQ bindings. For example, you can pause or resume an endpoint associated with one of those bindings.

About this task

The import or export must be installed as part of a Service Component Architecture (SCA) application in your server profile.

Note: This procedure applies only to Version 7 (or later) applications deployed to the runtime environment.

Procedure

1. Select the SCA module. From the administrative console, click **Applications** > **SCA Modules** and then click the *modulename*.
2. Under **Module components**, expand **Imports** or **Exports**.
3. Expand the import or export and then expand **Binding**. Make sure that you select one of the following bindings:
   - JMS
   - WebSphere MQ JMS
   - WebSphere MQ
4. Click on the binding to be administered.
5. To view or change the state of a binding endpoint, perform the following tasks:
   a. Click the **Runtime** tab.
   b. In the **Receiving Endpoints** table, select the check box for the endpoint.
   c. Click **Pause** or **Resume** to temporarily stop or restart the endpoint.

Results

The endpoint is paused or resumed.

Migrating WebSphere MQ Bindings from version 6 to version 7:

Migration is only required for WebSphere MQ bindings that contain pre-configured resources.

Specifying an activation specification

In WebSphere ESB version 7, the WebSphere MQ binding uses the WebSphere MQ resource adapter to receive messages, which requires an activation specification. If a WebSphere MQ binding has pre-configured WebSphere MQ resources, you need to define an additional activation specification JNDI name in the end-point configuration of the binding. This JNDI name must refer to an existing activation specification JMS resource on the server.

Modifying connection factory properties

Pre-configured connection factories need these custom properties removed:

- SENDEXIT
- RECEXIT
- SENDEXITINIT

- RECEXITINIT

Modifying destination properties

Pre-configured destinations need these custom properties added:

Table 36. Custom properties for WebSphere MQ queue destinations

Destination Type	Property Name	Property Value	Property Type
Send destination	MDWRITE	YES	java.lang.String
	MSGBODY	MQ	java.lang.String
	MDMSGCTX	SET_ALL_CONTEXT	java.lang.String
Receive destination	MDREAD	YES	java.lang.String
	MSGBODY	MQ	java.lang.String
	MDMSGCTX	SET_ALL_CONTEXT	java.lang.String

Administering service modules in Business Space

Use the administration widgets to browse deployed modules in a cell, view detailed information about each module, and edit module artifacts. If you have installed the Solution Administration template, these widgets are available in various combinations on the template pages.

A graphical view visually represents the services exposed and invoked by a module with appropriate contextual information about each. Detailed information about the module is also available and includes the module version, cell identifier (used to uniquely identify a versioned module in a cell), and endpoints. The graphical view provides overall module status and highlights any existing configuration problems.

The Module Browser widget is used with other administration widgets. It enables you to list the names of any properties and policies defined for the module. In addition, if you are using IBM Process Server, it lists the business rules, BPEL processes, business state machines, human tasks, business calendars, and security roles defined for the module. Select an artifact from the list in this widget to perform any necessary administration tasks in related widgets.

As a group, these widgets provide visibility into all the administrative artifacts in an application cluster, effectively aggregating solution administration. The data provided in these widgets can help you answer questions such as:

- What services are consumed in or exposed by a module?
- Is the module wiring correct, and are the correct versions of modules deployed?
- Are all service exports and service imports bound to existing modules?
- What is the status of a module? Is it running, or stopped?
- Are there any failed events in the module?
- What properties does the module expose?
- What mediation policies are associated with the module?

Browsing and administering modules:

Use the Administration widgets to view and administer all deployed SCA modules in a cell.

Before you begin

- If you are using a deployment environment, ensure the deployment manager must be running before you use the widget. In addition, the node agents must be running in order to see module status in the widget.
- If you have turned on security in Business Space, make sure you are using both administrative security and application security. For more information, see Setting up security for Business Space.
- If you want to display security roles for a module, make sure administrative security and application security are enabled.
- If you are using Microsoft Internet Explorer as your browser, configure it to display new versions of pages instead of cached versions. See Caching problem in default configuration of Microsoft Internet Explorer Versions 6, 7, and 8 for more information.

Required security role for this task: If administrative security is enabled, you must be logged in with an administrative role (Administrator, Configurator, Operator, Monitor, AdminsecurityManager, Deployer, or icsadmins) to view and administer modules.

About this task

For each module in the cell, you can see a list of artifacts associated with it in the Module Browser widget, including service exports, service imports, properties, policies, and service control points. If you are using IBM Process Server, you can also see BPEL processes, human tasks, business rules, business state machines, business calendars and security roles for the module.

The related Module Administration widget displays a graphical representation of module service exports and service imports and provides administrative access to module artifacts.

To browse for a module and examine its details, perform the following steps.

Procedure

1. Log into Business Space and open the page that contains the Module Browser and Module Administration widgets. The Module Browser widget lists all of the modules currently deployed to the cell.
2. Optional: If you want to display only a subset of modules, complete the following steps.
   a. From the menu on the widget toolbar, click **Edit Settings**.
   b. Select **Show selected modules**, and click **Add**.
   c. In the Select Modules to Add dialog box, select the names of modules that you want to show in the Module Browser widget. In the **Search modules** text box, you can type the first few letters of the name of a module, and the list of modules in the dialog box is refreshed with only the modules that begin with those letters.
   d. After selecting all modules that you want to show in the Module Browser widget, click **Add**. The Module Browser widget displays the specific modules you selected.
3. Browse the list of modules to locate the one you want to see in detail.
4. Click the arrow next to the module name to expand the tree view. The widget displays the following information about the module:
   - Service imports and service exports in the module

- Whether properties and policies have been defined for the module
- Service control points defined for the module

5. To see additional details about module artifacts, perform one of the following actions.

Option	Description
To view service imports and service exports	1. Click the name of the module to display a diagram of all of its service imports and service exports in the Module Administration widget. 2. Click any service import or export in the diagram to display more information. You can also click the name of an individual import or export in Module Browser to display its high-level binding endpoint details in the Module Administration widget.
To view or administer module policies	1. Click the Module Policies section of the browser tree to view the policy attachments defined for the module in the Module Administration widget. 2. Optionally, create new policy attachments and click **Save**.
To view or administer module properties	1. Click the Module Properties section of the browser tree to list all promoted properties defined for the module in the Module Administration widget. 2. Optionally, modify property values and click **Save**.
To view all of the service control points for a module and display details for a service control point	To view all of the service control points for a module: 1. Expand the Service Control Points section of the browser tree to list all service control points. 2. Click Service Control Points to display all service control points in the Module Administration widget. To display details for a service control point: 1. Expand the Service Control Points section of the browser tree to list all service control points. 2. Click the name of a service control point to display its details in the Module Administration widget.

Administering mediation policies for modules:

Use the Mediation Policy Administration widget to work with *mediation policies* associated with modules. Using mediation policies you can dynamically control service interactions, using contextual information.

Before you begin

If you want to work with mediation policies associated with modules, then create a business space using the Solution Administration template. The Solution Administration template creates a business space from which you can view and administer modules.

About this task

Depending on whether your mediation policies are associated with modules or with target services, the Mediation Policy Administration widget interacts with different widgets.

- If your mediation policies are associated with modules, the Mediation Policy Administration widget can interact with the Module Browser widget and the Module Administration widget.
- If your mediation policies are associated with target services, the Mediation Policy Administration widget can interact with the Service Browser widget.

Procedure

1. Log on to Business Space.
2. In your Solution Administration business space, open a Module Administration page.

Results

The initial widget details are displayed.

The Module Browser widget lists the modules installed on the application server. The application server can be either WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Process Server.

The content of the Module Administration widget depends on what you select in the Module Browser widget. If you select **Module Policies**, the Module Administration widget displays the Mediation Policy Administration widget.

What to do next

Display, create, attach, or delete mediation policies.

Modules: Displaying mediation policies for modules:

Use the Mediation Policy Administration widget, to create *mediation policies* and policy attachments in WebSphere Service Registry and Repository (WSRR). Using mediation policies you can control service interactions, using contextual information.

Before you begin

1. Use IBM Integration Designer to create a module containing a Policy Resolution mediation primitive.
2. Deploy the module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.

3. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
4. Load the enterprise archive (EAR) file, containing your module, into WSRR.
5. Create a business space that contains the Module Browser widget and the Mediation Policy Administration widget.

About this task

You can control service requests dynamically by using mediation policies to override module properties at run time. Such mediation policies are stored in WSRR. You can define one or more mediation policies for your module, and each mediation policy can override one or more module properties. Optionally, you can create one or more gate conditions on each policy attachment. When service requests are processed, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated mediation policy can be used.

Procedure

1. Log on to Business Space.
2. Open the business space that you created for administering mediation policies associated with modules.
3. From the Module Browser widget, select **Mediation Policies**. The Mediation Policy Administration widget is refreshed. If there are existing policy attachments they are displayed.
4. From the Mediation Policy Administration widget, if you have more than one WSRR definition, then select the definition used by your module. If you change the WSRR definition, the list of policy attachments changes.
5. Click the **Edit** icon of the policy attachment that you want to work with. Each policy attachment row has a pencil icon, at the end of the row, that you can click to view mediation policy information.

Results

The Mediation Policy Administration displays the following information:

- Assertions: The module properties that the mediation policy can override. In WSRR, the module properties appear as policy assertions.
  - Group Name: The group to which the property belongs. By default, the group name is the name of the mediation flow component.
  - Property Name: The alias name of the property. The alias name identifies the property in the mediation flow.
  - Value: The current value in the mediation policy, rather than the current value in the module. When a mediation policy is available and suitable, the mediation policy value takes precedence.
- Gate Conditions (Optional): Conditions that must be met before the mediation policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.
  - Name: The name of a gate condition is always prefixed with the string *medGate_*.
  - Value: The value of the gate condition, for example, `country = France` or `Age > 59`.

Modules: Creating mediation policies for modules:

Use the Mediation Policy Administration widget, to create *mediation policies* and policy attachments in WebSphere Service Registry and Repository (WSRR). Using mediation policies you can control service interactions, using contextual information.

Before you begin

1. Use IBM Integration Designer to create a module containing a Policy Resolution mediation primitive.
2. Deploy the module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
4. Load the enterprise archive (EAR) file, containing your module, into WSRR.
5. Create a business space that contains the Module Browser widget and the Mediation Policy Administration widget.

About this task

You can control service requests dynamically by using mediation policies to override module properties at run time. Such mediation policies are stored in WSRR. You can define one or more mediation policies for your module, and each mediation policy can override one or more module properties. Optionally, you can create one or more gate conditions on each policy attachment. When service requests are processed, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated mediation policy can be used.

Procedure

1. Log on to Business Space.
2. Open the business space that you created for administering mediation policies associated with modules.
3. From the Module Browser widget, select **Mediation Policies**. The Mediation Policy Administration widget is refreshed. If there are existing policy attachments they are displayed.
4. If you have more than one WSRR definition, select the definition used by your module.
5. Enter the name of the **New policy attachment**. Mediation policy attachments associate a mediation policy with a module. In WSRR, the mediation policy and policy attachment are separate objects.
6. Click **Create...** The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of the new mediation policy.
7. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
8. Enter a name in the **New policy** field. This is the name of the mediation policy you want to create and attach to the module.
9. Click **Next...** The Mediation Policy Administration widget is refreshed. You can now add assertions and gate conditions.

   Note: Using Business Space, you cannot edit assertions after you create a mediation policy. Therefore, you must add all the assertions that you require before you save the mediation policy.

10. Create one or more assertions. Assertions are module properties that the policy can override. In WSRR, the module properties that you want to override appear as policy assertions.

    Note: The widget requires each policy attachment to have at least one assertion.

    a. Select a **Property name**. The name is the alias name of the property. The alias name identifies the property in the mediation flow.
    b. Enter a suitable value in the **Value** field; for example, `All` or `10` or `/body/input/address`. When available, the policy value takes precedence at run time. If a policy is not found, or is unsuitable, the runtime environment uses the promoted property value.
    c. Click **Add Assertion**.
11. Optional: Create one or more gate conditions. Gate conditions must be met before the policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.
    a. Enter a gate condition name in the **Gate condition name** field. The name of a gate condition is always prefixed with the string `medGate_`.
    b. Enter a gate condition value in the **Value** field. The gate condition value is made up of the following parts: *policy condition name*, *operation* and *gate value*.
       - The *policy condition name* you enter must map to a **`Policy condition name`** in the module.
       - The *operation* can be: `=` , `!=` , `>` , `<` , `<=` or `>=` .
       - The *gate value* is the value being compared, for example, `country = France`.
    c. Click **Add Gate Condition**.
12. Optional: If you want to delete an assertion or gate condition, click the delete icon of the appropriate assertion or gate condition. If you hover over an assertion or gate condition, the delete icon, a cross, appears at the end of the row.
13. Click **Save**.

Results

In WSRR, a mediation policy and a policy attachment are created. The policy attachment connects the mediation policy to the module. If you added a gate condition, WSRR creates a user property on the policy attachment. The user property represents the condition.

Modules: Attaching existing mediation policies to modules:

Use the Mediation Policy Administration widget, to attach existing *mediation policies* to modules. Using mediation policies you can control service interactions, using contextual information.

Before you begin

1. Use IBM Integration Designer to create a module containing a Policy Resolution mediation primitive.
2. Deploy the module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.

3. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
4. Load the enterprise archive (EAR) file, containing your module, into WSRR.
5. Create a business space that contains the Module Browser widget and the Mediation Policy Administration widget.

About this task

You can control service requests dynamically by using mediation policies to override module properties at run time. Such mediation policies are stored in WSRR. You can define one or more mediation policies for your module, and each mediation policy can override one or more module properties. Optionally, you can create one or more gate conditions on each policy attachment. When service requests are processed, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated mediation policy can be used.

Procedure

1. Log on to Business Space.
2. Open the business space that you created for administering mediation policies associated with modules.
3. From the Module Browser widget, select **Mediation Policies**. The Mediation Policy Administration widget is refreshed. If there are existing policy attachments they are displayed.
4. If you have more than one WSRR definition, select the definition used by your module.
5. Enter the name of the **New policy attachment**. Mediation policy attachments associate a mediation policy with a module. In WSRR, the mediation policy and policy attachment are separate objects.
6. Click **Create...** The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of an existing mediation policy for that group.
7. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
8. Click **Use existing**.
9. Select a mediation policy from the **Select a policy** menu. The mediation policies displayed depend on the group you selected.
10. Click **Next...** The Mediation Policy Administration widget is refreshed. You can now add gate conditions.

    Note: Using Business Space, you cannot edit mediation policy assertions after you create a mediation policy. However, because gate conditions exist on the mediation policy attachment, you can add gate conditions when you create a new policy attachment.
11. Optional: Define one or more gate conditions. Gate conditions must be met before the policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.
    a. Enter a gate condition name in the **Gate condition name** field. The name of a gate condition is always prefixed with the string `medGate_`.
    b. Enter a gate condition value in the **Value** field. The gate condition value is made up of the following parts: *policy condition name*, *operation* and *gate value*.

- The *policy condition name* you enter must map to a **Policy condition name** in the module.
- The *operation* can be: `=` , `!=` , `>` , `<` , `<=` or `>=` .
- The *gate value* is the value being compared, for example, `country = France`.

c. Click **Add Gate Condition**.

12. Click **Save**.

Results

In WSRR, a policy attachment is created that associates the selected mediation policy with the selected module.

The Mediation Policy Administration widget is refreshed, and the new policy attachment is added to the list of policy attachments.

Modules: Deleting mediation policies for modules:

Use the Mediation Policy Administration widget, to delete policy attachments and associated *mediation policies* in WebSphere Service Registry and Repository (WSRR).

Before you begin

1. Use IBM Integration Designer to create a module containing a Policy Resolution mediation primitive.
2. Deploy the module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
4. Load the enterprise archive (EAR) file, containing your module, into WSRR.
5. Create a business space that contains the Module Browser widget and the Mediation Policy Administration widget.

About this task

You can control service requests dynamically by using mediation policies to override module properties at run time. Such mediation policies are stored in WSRR. You can define one or more mediation policies for your module, and each mediation policy can override one or more module properties. Optionally, you can create one or more gate conditions on each policy attachment. When service requests are processed, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated mediation policy can be used.

Procedure

1. Log on to Business Space.
2. Open the business space that you created for administering mediation policies associated with modules.
3. From the Module Browser widget, select **Mediation Policies**. The Mediation Policy Administration widget is refreshed. If there are existing policy attachments they are displayed.
4. If you have more than one WSRR definition, then select the definition used by your module. If you change the WSRR definition, the list of policy attachments changes.

5. Click the **Delete** icon of the policy attachment that you want to delete. Each policy attachment row has a cross icon, at the end of the row, that you can click to delete the policy attachment. You are asked if you want to delete the policy attachment.
6. Click **OK**.

Results

In WSRR, the policy attachment is deleted. Any associated mediation policies are also deleted, unless they are being used by another policy attachment.

Administering services in Business Space

Use the administration widgets to list services defined in WebSphere Service Registry and Repository (WSRR) and administer the mediation policies associated with the services.

Use the Service Browser widget to list services defined in WSRR. In addition, use the Service Browser widget to list the endpoints and operations defined for the services.

Use the Mediation Policy Administration widget to administer mediation policies associated with services.

The data provided in these widgets can help you answer questions such as:

- What services are defined in an instance of WSRR?
- What endpoints and operations are defined for a service?
- What mediation policies are associated with a service?

Listing services defined in WSRR:

Use the Service Browser widget to list services defined in an instance of WebSphere Service Registry and Repository (WSRR).

Before you begin

1. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
2. Load the WSDL file that defines your service into WSRR.
3. Create a business space using the Service Administration template. The Service Administration template creates a business space that contains the Service Browser widget.

About this task

You can list the services that are defined in an instance of WSRR. In addition, you can list the endpoints and operations that are defined for the services.

Procedure

1. Log on to Business Space.
2. In your Service Administration business space, open a Service Administration page.
3. From the Service Browser widget, check that the correct WSRR definition is displayed. If the correct WSRR definition is not displayed, select the correct WSRR definition. If your application server has definitions for more than one instance of WSRR, you can display the services that are defined on each WSRR.

Results

When you change the WSRR definition, the list of services is refreshed.

What to do next

Display the names of the endpoints and operations for a service. Select a service and expand the tree structure under **Mediation Policies**.

Note: Icons are used to indicate whether a name refers to a service, endpoint, or operation. In addition, when you select a level under **Mediation Policies**, the Mediation Policy Administration widget displays the level.

Administering mediation policies for services:

Use the Mediation Policy Administration widget to work with *mediation policies* associated with target services. Using mediation policies you can dynamically control service interactions, using contextual information.

Before you begin

If you want to work with mediation policies associated with target services, then create a business space using the Service Administration template. The Service Administration template creates a business space from which you can view and administer services.

About this task

Depending on whether your mediation policies are associated with modules or with target services, the Mediation Policy Administration widget interacts with different widgets.

- If your mediation policies are associated with modules, the Mediation Policy Administration widget can interact with the Module Browser widget and the Module Administration widget.
- If your mediation policies are associated with target services, the Mediation Policy Administration widget can interact with the Service Browser widget.

Procedure

1. Log on to Business Space.
2. In your Service Administration business space, open a Service Administration page.

Results

The initial widget details are displayed.

The Service Browser widget lists the services defined on a particular instance of WebSphere Service Registry and Repository (WSRR).

The contents of the Mediation Policy Administration widget depends on what you select in the Service Browser widget.

What to do next

Display, create, attach, or delete mediation policies.

Services: Displaying mediation policies for services:

Use the Mediation Policy Administration widget, to display *mediation policies* and policy attachments, which exist in WebSphere Service Registry and Repository (WSRR). Using mediation policies you can control service interactions, using contextual information.

Before you begin

1. Use IBM Integration Designer to create a module containing a Policy Resolution mediation primitive.
2. Deploy the module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
4. In WSRR, load the enterprise archive (EAR) file containing your module. Also, load the WSDL documents for the services that you want to attach mediation policies to.
5. Create a business space using the Service Administration template. The Service Administration template creates a business space that contains the widgets you need.

About this task

You can control service requests dynamically by using mediation policies to override module properties at run time. Such mediation policies are stored in WSRR. You can define one or more mediation policies for services used by your module, and each mediation policy can override one or more module properties. Optionally, you can create one or more gate conditions on each policy attachment. When service requests are processed, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated mediation policy can be used.

Procedure

1. Log on to Business Space.
2. In your Service Administration business space, open a Service Administration page.
3. From the Service Browser widget, check that the correct WSRR definition is displayed. If the correct WSRR definition is not displayed, select the correct WSRR definition. If your application server has definitions for more than one instance of WSRR, you can display the services that are defined on each WSRR. When you change the WSRR definition, the list of services is refreshed.
4. Select the level for which you want to display policy attachments. Mediation policies can be attached at the level of the service, endpoint, or operation. The Mediation Policy Administration widget is refreshed. If there are existing policy attachments they are displayed.
5. From the Mediation Policy Administration widget, click the **Edit** icon of the policy attachment that you want to work with. Each policy attachment row has a pencil icon, at the end of the row, that you can click to view mediation policy information.

Results

The Mediation Policy Administration displays the following information:

- Assertions: The module properties that the mediation policy can override. In WSRR, the module properties appear as policy assertions.
  - Group Name: The group to which the property belongs. By default, the group name is the name of the mediation flow component.
  - Property Name: The alias name of the property. The alias name identifies the property in the mediation flow.
  - Value: The current value in the mediation policy, rather than the current value in the module. When a mediation policy is available and suitable, the mediation policy value takes precedence.
- Gate Conditions (Optional): Conditions that must be met before the mediation policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.
  - Name: The name of a gate condition is always prefixed with the string *medGate_*.
  - Value: The value of the gate condition, for example, `country = France` or `Age > 59`.

Services: Creating mediation policies for services:

Use the Mediation Policy Administration widget, to create *mediation policies* and policy attachments in WebSphere Service Registry and Repository (WSRR). Using mediation policies you can control service interactions, using contextual information.

Before you begin

1. Use IBM Integration Designer to create a module containing a Policy Resolution mediation primitive.
2. Deploy the module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
4. In WSRR, load the enterprise archive (EAR) file containing your module. Also, load the WSDL documents for the services that you want to attach mediation policies to.
5. Create a business space using the Service Administration template. The Service Administration template creates a business space that contains the widgets you need.

About this task

You can control service requests dynamically by using mediation policies to override module properties at run time. Such mediation policies are stored in WSRR. You can define one or more mediation policies for services used by your module, and each mediation policy can override one or more module properties. Optionally, you can create one or more gate conditions on each policy attachment. When service requests are processed, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated mediation policy can be used.

Procedure

1. Log on to Business Space.
2. In your Service Administration business space, open a Service Administration page.
3. From the Service Browser widget, if the correct WSRR definition is not displayed select the correct WSRR definition. If your application server has definitions for more than one instance of WSRR, you can display the services that are defined on each WSRR. The list of services is refreshed.
4. Select the level at which you want to create a mediation policy. You can attach a mediation policy at the level of the service, endpoint, or operation. The Mediation Policy Administration widget is refreshed. The following information is displayed:
   - The name of the service, endpoint, or operation that you selected.
   - The WSRR definition that you selected.
   - Any policy attachments that exist for the service, endpoint, or operation that you selected.
5. Enter the name of the **New policy attachment**. Policy attachments associate a mediation policy with a target service. In WSRR, the mediation policy and the policy attachment are separate objects.
6. Click **Create...** The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of the new mediation policy.
7. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
8. Enter a name in the **New policy** field. This is the name of the mediation policy you want to create and attach to the service, endpoint, or operation.
9. Click **Next...** The Mediation Policy Administration widget is refreshed. You can now add assertions and gate conditions.

   Note: Using Business Space, you cannot edit assertions after you create a mediation policy. Therefore, you must add all the assertions that you require before you save the mediation policy.
10. Define one or more assertions. Assertions are module properties that the mediation policy can override. In WSRR, the module properties that you want to override appear as policy assertions.

    Note: The widget requires each policy attachment to have at least one assertion.

    a. Select a **Property name**. The name is the alias name of the property. The alias name identifies the property in the mediation flow.
    b. Enter a suitable value in the **Value** field; for example, `All` or `10` or `/body/input/address`. When available, the policy value takes precedence at run time. If a policy is not found, or is unsuitable, the runtime environment uses the promoted property value.
    c. Click **Add Assertion**.
11. Optional: Define one or more gate conditions. Gate conditions must be met before the policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.

    a. Enter a gate condition name in the **Gate condition name** field. The name of a gate condition is always prefixed with the string `medGate_`.

b. Enter a gate condition value in the **Value** field. The gate condition value is made up of the following parts: *policy condition name*, *operation* and *gate value*.
   - The *policy condition name* you enter must map to a **`Policy condition name`** in the module.
   - The *operation* can be: `=` , `!=` , `>` , `<` , `<=` or `>=` .
   - The *gate value* is the value being compared, for example, `country = France`.

c. Click **Add Gate Condition**.

12. Optional: If you want to delete an assertion or gate condition, click the delete icon of the appropriate assertion or gate condition. If you hover over an assertion or gate condition, the delete icon, a cross, appears at the end of the row.
13. Click **Save**.

Results

In WSRR, a mediation policy and a policy attachment are created. The policy attachment connects the mediation policy to the service, endpoint, or operation. If you added a gate condition, WSRR creates a user property on the policy attachment. The user property represents the condition.

Services: Attaching existing mediation policies to services:

Use the Mediation Policy Administration widget, to attach existing *mediation policies* to service endpoints. Using mediation policies you can control service interactions, using contextual information.

Before you begin

1. Use IBM Integration Designer to create a module containing a Policy Resolution mediation primitive.
2. Deploy the module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
4. In WSRR, load the enterprise archive (EAR) file containing your module. Also, load the WSDL documents for the services that you want to attach mediation policies to.
5. Create a business space using the Service Administration template. The Service Administration template creates a business space that contains the widgets you need.

About this task

You can control service requests dynamically by using mediation policies to override module properties at run time. Such mediation policies are stored in WSRR. You can define one or more mediation policies for services used by your module, and each mediation policy can override one or more module properties. Optionally, you can create one or more gate conditions on each policy attachment. When service requests are processed, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated mediation policy can be used.

Procedure

1. Log on to Business Space.
2. In your Service Administration business space, open a Service Administration page.
3. From the Service Browser widget, if the correct WSRR definition is not displayed select the correct WSRR definition. If your application server has definitions for more than one instance of WSRR, you can display the services that are defined on each WSRR. The list of services is refreshed.
4. Select the level at which you want to attach a mediation policy. You can attach a mediation policy at the level of the service, endpoint, or operation. The Mediation Policy Administration widget is refreshed. The following information is displayed:
   - The name of the service, endpoint, or operation that you selected.
   - The WSRR definition that you selected.
   - Any policy attachments that exist for the service, endpoint, or operation that you selected.
5. Enter the name of the **New policy attachment**. Policy attachments associate a mediation policy with a target service. In WSRR, the mediation policy and the policy attachment are separate objects.
6. Click **Create...** The Mediation Policy Administration widget is refreshed. You can now specify the group of properties you want to work with, and the name of an existing mediation policy for that group.
7. Select a **Group name**. Each group contains module properties. Select the group whose property values you want to override.
8. Click **Use existing**.
9. Select a mediation policy from the **Select a policy** menu. The mediation policies displayed depend on the group you selected. Because a target service might be called from different modules, the mediation policy associated with the target service might not affect the service request. A mediation policy can only affect a service request if the service, endpoint, or operation that you selected is called by an appropriate module. An appropriate module is one that has properties that the mediation policy can override.
10. Click **Next...** The Mediation Policy Administration widget is refreshed. You can now add gate conditions.

    Note: Using Business Space, you cannot edit mediation policy assertions after you create a mediation policy. However, because gate conditions exist on the mediation policy attachment, you can add gate conditions when you create a new policy attachment.
11. Optional: Define one or more gate conditions. Gate conditions must be met before the policy can be used. In WSRR, gate conditions are user properties on the policy attachment object.
    a. Enter a gate condition name in the **Gate condition name** field. The name of a gate condition is always prefixed with the string `medGate_`.
    b. Enter a gate condition value in the **Value** field. The gate condition value is made up of the following parts: *policy condition name*, *operation* and *gate value*.
       - The *policy condition name* you enter must map to a **`Policy condition name`** in the module.
       - The *operation* can be: `=` , `!=` , `>` , `<` , `<=` or `>=` .

- The *gate value* is the value being compared, for example, `country = France`.

c. Click **Add Gate Condition**.

12. Click **Save**.

Results

In WSRR, a policy attachment is created that associates the selected mediation policy with the selected service, endpoint, or operation.

The Mediation Policy Administration widget is refreshed, and the new policy attachment is added to the list of policy attachments.

Services: Deleting mediation policies for services:

Use the Mediation Policy Administration widget, to delete policy attachments and associated *mediation policies* in WebSphere Service Registry and Repository (WSRR).

Before you begin

1. Use IBM Integration Designer to create a module containing a Policy Resolution mediation primitive.
2. Deploy the module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Ensure that WebSphere ESB or IBM Business Process Manager have a definition for the WSRR that you want to use.
4. In WSRR, load the enterprise archive (EAR) file containing your module. Also, load the WSDL documents for the services that you want to attach mediation policies to.
5. Create a business space using the Service Administration template. The Service Administration template creates a business space that contains the widgets you need.

About this task

You can control service requests dynamically by using mediation policies to override module properties at run time. Such mediation policies are stored in WSRR. You can define one or more mediation policies for services used by your module, and each mediation policy can override one or more module properties. Optionally, you can create one or more gate conditions on each policy attachment. When service requests are processed, gate conditions are compared to the condition values in the message. All the gate conditions must be met before an associated mediation policy can be used.

Procedure

1. Log on to Business Space.
2. In your Service Administration business space, open a Service Administration page.
3. From the Service Browser widget, if the correct WSRR definition is not displayed select the correct WSRR definition. If your application server has definitions for more than one instance of WSRR, you can display the services that are defined on each WSRR. The list of services is refreshed.
4. Select the level at which you want to delete a policy attachment and associated mediation policies. You can attach a mediation policy at the level of the service,

endpoint, or operation. The Mediation Policy Administration widget is refreshed. The following information is displayed:
   - The name of the service, endpoint, or operation that you selected.
   - The WSRR definition that you selected.
   - Any policy attachments that exist for the service, endpoint, or operation that you selected.
5. Click the delete icon of the policy attachment that you want to delete. Each policy attachment row has a cross icon, at the end of the row, that you can click to delete the policy attachment. You are asked if you want to delete the policy attachment.
6. Click **OK**.

Results

In WSRR, the policy attachment is deleted. Any associated mediation policies are also deleted, unless they are being used by another policy attachment.

Administering services using proxy gateways:

Use the Proxy Gateway widget to administer *proxy groups* defined for a *proxy gateway*. A proxy gateway is a module that receives Web service requests and forwards them to an endpoint defined in a proxy group.

Before you begin

Create a business space using the Service Administration template. The Service Administration template creates a Proxy Gateway page, containing the Proxy Gateway widget.

About this task

Proxy groups must be defined in the following places:
- In the proxy gateway module, using IBM Integration Designer.
- In the built-in configuration store, in WebSphere ESB or IBM Process Server. When you deploy a proxy gateway module to WebSphere ESB or IBM Process Server, any new proxy groups defined by the module are automatically created.

  Note: When you uninstall a proxy gateway module, the proxy groups defined by the module are not automatically deleted.

Proxy groups contain *virtual services* that you map to real service endpoints, using the Proxy Gateway widget.

Procedure
1. Log on to Business Space.
2. To work with proxy gateways and proxy groups, open the Proxy Gateway page.

Results

Initially, the Proxy Gateway widget displays the following information:
- Proxy Group: A group to which you can add virtual services that you map to real service endpoints. You can use the icon after the Proxy Group heading to sort the entries in ascending (^) or descending (v) order.

- Proxy Gateway: A module that can forward Web service requests to an endpoint defined in a proxy group. You can use the icon after the Proxy Gateway heading to sort the entries in ascending (^) or descending (v) order.

What to do next

Display the contents of proxy groups. Update or delete proxy groups.

Displaying proxy groups:

Use the Proxy Gateway widget to list your *proxy gateways* and *proxy groups*. Also, use the Proxy Gateway widget to display the details of a particular proxy group.

Before you begin

1. Use IBM Integration Designer to create a proxy gateway. Use the wizard to create a proxy gateway module, which is an SCA module containing a Gateway Endpoint Lookup mediation primitive.
2. Deploy the proxy gateway module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Create a business space using the Service Administration template. The Service Administration template creates a Proxy Gateway page, containing the Proxy Gateway widget.

About this task

A proxy gateway is a module that receives Web service requests and forwards them to an endpoint defined in a proxy group. Proxy groups contain *virtual services* that you map to real service endpoints. A virtual service can have one or more endpoints associated with it.

Procedure

1. Log on to Business Space.
2. Open a Proxy Gateway page. The Proxy Gateway widget lists the proxy gateways and proxy groups that are installed on your system.
3. From the Proxy Gateway widget, select the **Proxy Group** whose details you want to display. Click the pencil icon at the end of the relevant row. The Proxy Gateway widget is refreshed.

Results

If the configuration store contains virtual services for the proxy group, the Proxy Gateway widget displays the following information:

- Enabled: An icon indicates whether you can send messages to the virtual service. On another panel, you can enable or disable virtual services.
- Virtual Service: The name of a proxy service. You can associate a virtual service with one or more existing service endpoints.
- Endpoint: One or more existing service endpoints that are associated with the virtual service.

What to do next

Work with an existing virtual service or add a new virtual service.

Adding virtual services to proxy groups:

Use the Proxy Gateway widget to add a *virtual service* to a *proxy group*.

Before you begin

1. Use IBM Integration Designer to create a proxy gateway. Use the wizard to create a proxy gateway module, which is an SCA module containing a Gateway Endpoint Lookup mediation primitive.
2. Deploy the proxy gateway module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Create a business space using the Service Administration template. The Service Administration template creates a Proxy Gateway page, containing the Proxy Gateway widget.

About this task

A proxy gateway is a module that receives Web service requests and forwards them to an endpoint defined in a proxy group. Proxy groups contain virtual services that you map to real service endpoints. A virtual service can have one or more endpoints associated with it.

When you create your proxy gateway module, you define the proxy groups for the module. After you deploy your proxy gateway module, you can use the Proxy Gateway widget to add virtual services to the proxy groups. The virtual service information is stored in the built-in configuration store that exists in WebSphere ESB and IBM Business Process Manager.

When the proxy gateway processes a client request, the virtual service name used to look up the endpoints must match the virtual service name in the client request. If you create a proxy gateway module with the default type of routing, which is URL-based, and use the URL available in the resolved WSDL, then the routing of the request occurs automatically. If you create a proxy gateway module with XPath-based routing, ensure that the message location you specify contains the correct virtual service name.

Procedure

1. Log on to Business Space.
2. Open a Proxy Gateway page.
3. From the Proxy Gateway widget, select the **Proxy Group** that you want to work with. Click the pencil icon at the end of the relevant row. The Proxy Gateway widget is refreshed. If the configuration store contains virtual services for the proxy group, the virtual services are displayed. If a virtual service is associated with more than one endpoint, only the first endpoint is displayed in the table.
4. Add a virtual service.
   a. Enter the location of the WSDL that describes the virtual service. The WSDL might be stored in WebSphere Service Registry and Repository (WSRR) or another repository. When the widget is refreshed, the service name in the WSDL is used to populate the **Virtual Service Name** field.
   b. Click **Add Service...** The Proxy Gateway widget is refreshed and shows the following information:

      Note: If your WSDL document describes multiple WSDL services, the Proxy Gateway widget imports the first WSDL service only.

      - Port Type: The WSDL portType of the virtual service.

- **Virtual Service Name**: The name of the virtual service that is stored in the configuration store. The default name is entered for you, and is based on the service name in the WSDL you specified.
- Virtual Service URLs:
  - Proxy Gateway: The name of the proxy gateway module.
  - Endpoint: The endpoint of the virtual service.
- **Enable Virtual Service**: A check box that indicates whether you can send messages to the virtual service. By default, virtual services are enabled.
- **Endpoint URLs**: One or more network-addressable endpoints to which a message can be forwarded. If you define a list of endpoints, you can determine the order in which the services are tried. You determine the order by moving the endpoints up and down in the list.
- Advanced Service Properties: If you need to do special processing, for a particular virtual service, you might use the Advanced Service Properties. The Advanced Service Properties are key-value pairs that you want to be accessible in the mediation flow, after the Gateway Endpoint Lookup mediation primitive. For example, you might want to specify the location of the XSL style sheet that relates to this virtual service. At run time, the key-value pairs are stored in the EndpointLookupContext of the service message object (SMO).
  - **Name**: The name of the key.
  - **Value**: The value of the key.

5. Optional: Add another endpoint to the virtual service. Click **Add Endpoint.**
6. Optional: Delete one of the endpoints, defined for the virtual service. Click the cross icon at the end of the relevant row.
7. Save the endpoint information. Click **Save**.

   Note: To return to the previous screen, click **Cancel**.

Results

The new virtual service is added to the built-in configuration store, and the Proxy Gateway widget is refreshed.

What to do next

1. Retrieve the WSDL that your client uses to call a virtual service. You can obtain the WSDL by entering the endpoint of the virtual service URL in a Web Browser, and appending the string: `?wsdl`. For example, `http://zzz/Gold?wsdl`, where `http://zzz/` is the address of the proxy gateway and `Gold` is the name of the virtual service.

   Note: The endpoint of the virtual service URL is specified in the Endpoint field, under the Virtual Service URLs heading.
2. Use your client to access the proxy gateway. The proxy gateway routes your request to the real service associated with the virtual service.

Adding endpoints to virtual services:

Use the Proxy Gateway widget to add endpoints to *virtual services* that belong to a *proxy group*.

Before you begin

1. Use IBM Integration Designer to create a proxy gateway. Use the wizard to create a proxy gateway module, which is an SCA module containing a Gateway Endpoint Lookup mediation primitive.
2. Deploy the proxy gateway module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Create a business space using the Service Administration template. The Service Administration template creates a Proxy Gateway page, containing the Proxy Gateway widget.

About this task

A proxy gateway is a module that receives Web service requests and forwards them to an endpoint defined in a proxy group. Proxy groups contain virtual services that you map to real service endpoints. A virtual service can have one or more endpoints associated with it.

When you create your proxy gateway module, you define the proxy groups for the module. After you deploy your proxy gateway module, you can use the Proxy Gateway widget to add virtual services to the proxy groups. The virtual service information is stored in the built-in configuration store that exists in WebSphere ESB and IBM Business Process Manager.

When the proxy gateway processes a client request, the virtual service name used to look up the endpoints must match the virtual service name in the client request. If you create a proxy gateway module with the default type of routing, which is URL-based, and use the URL available in the resolved WSDL, then the routing of the request occurs automatically. If you create a proxy gateway module with XPath-based routing, ensure that the message location you specify contains the correct virtual service name.

Procedure

1. Log on to Business Space.
2. Open a Proxy Gateway page.
3. From the Proxy Gateway widget, select the **Proxy Group** that you want to work with. Click the pencil icon at the end of the relevant row. The Proxy Gateway widget is refreshed. If the configuration store contains virtual services for the proxy group, the virtual services are displayed. If a virtual service is associated with more than one endpoint, only the first endpoint is displayed in the table.
4. Select the virtual service that you want to work with. Click the pencil icon at the end of the relevant row. The Proxy Gateway widget is refreshed and shows the following information:
   - Port Type: The WSDL portType of the virtual service.
   - **Virtual Service Name**: The name of the virtual service that is stored in the configuration store. The default name is entered for you, and is based on the service name in the resolved WSDL.
   - Virtual Service URLs:
     - Proxy Gateway: The name of the proxy gateway module.
     - Endpoint: The endpoint of the virtual service.
   - **Enable Virtual Service**: A check box that indicates whether you can send messages to the virtual service.

- **Endpoint URLs**: One or more network-addressable endpoints to which a message can be forwarded. If you define a list of endpoints, you can determine the order in which the services are tried. You determine the order by moving the endpoints up and down in the list.
- Advanced Service Properties: If you need to do special processing, for a particular virtual service, you might use the Advanced Service Properties. The Advanced Service Properties are key-value pairs that you want to be accessible in the mediation flow, after the Gateway Endpoint Lookup mediation primitive. For example, you might want to specify the location of the XSL style sheet that relates to this virtual service. At run time, the key-value pairs are stored in the EndpointLookupContext of the service message object (SMO).
  - **Name**: The name of the key.
  - **Value**: The value of the key.

5. Add an endpoint to the virtual service. Click **Add Endpoint.**
6. Save the endpoint information. Click **Save.**

Results

The new endpoint is added to the built-in configuration store, and the Proxy Gateway widget is refreshed.

What to do next

1. Retrieve the WSDL that your client must use to call a virtual service. You can obtain the WSDL by entering the endpoint of the virtual service URL in a Web Browser, and appending the string: `?wsdl`. For example, `http://zzz/Gold?wsdl`, where `http://zzz/` is the address of the proxy gateway and `Gold` is the name of the virtual service.

   Note: The endpoint of the virtual service URL is specified in the Endpoint field, under the Virtual Service URLs heading.
2. Use your client to access the proxy gateway. The proxy gateway routes your request to the real service associated with the virtual service.

Deleting proxy groups from the built-in store:

Use the Proxy Gateway widget to delete *proxy groups* from the built-in store.

Before you begin

1. Use IBM Integration Designer to create a proxy gateway. Use the wizard to create a proxy gateway module, which is an SCA module containing a Gateway Endpoint Lookup mediation primitive.
2. Install the proxy gateway module to WebSphere Enterprise Service Bus (WebSphere ESB) or IBM Business Process Manager.
3. Create a business space using the Service Administration template. The Service Administration template creates a Proxy Gateway page, containing the Proxy Gateway widget.
4. Uninstall the proxy gateway module.

About this task

A proxy gateway is a module that receives Web service requests and forwards them to an endpoint defined in a proxy group.

When you create a proxy gateway module, you define the proxy groups for the module. When you install your proxy gateway module, if the proxy groups do not exist in the built-in configuration store, they are automatically created.

When you uninstall your proxy gateway module, the proxy groups are not automatically deleted from the built-in configuration store. You must use the Proxy Gateway widget to delete the proxy groups and associated virtual services.

Procedure

1. Log on to Business Space.
2. Open a Proxy Gateway page.
3. From the Proxy Gateway widget, delete a proxy group that has no associated proxy gateway. Click the cross icon at the end of the relevant row.

Results

The proxy group is removed from the built-in configuration store, and the Proxy Gateway widget is refreshed.

Administering enterprise applications

Use the console's Enterprise application page to view and administer enterprise applications installed on the server.

To view the values specified for an application's configuration, open the page (click **Applications > Application Types> WebSphere enterprise applications**) and select the application name from the list. The application details page opens and displays the application's configuration properties and, if appropriate, local topology. From this page, you can modify existing values and link to additional console pages for configuring the application.

To administer an enterprise application, select it by clicking the check box next to its name and then use one of the following buttons:

Table 37. Buttons for administering enterprise applications

Button	Resulting action
Start	Attempts to run the application. After the application starts successfully, the state of the application changes to one of the following: • `Started`: The application has started on all deployment targets • `Partial Start`: The application is still starting on one or more of the deployment targets
Stop	Attempts to stop the processing of the application. After the application stops successfully, the state of the application changes to one of the following: • `Stopped`: The application has stopped on all deployment targets • `Partial Stop`: The application is still stopping on one or more of the deployment targets
Install	Opens a wizard to help you deploy an enterprise application or module (such as a .jar, .war, or .sar or .rar file) onto a server or cluster.

Table 37. Buttons for administering enterprise applications (continued)

Button	Resulting action
Uninstall	Deletes the application from the product configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed after the configuration is saved and synchronized with the nodes.
Update	Opens a wizard to help you update application files deployed on a server. You can update the full application, a single module, a single file, or part of the application. If a new file or module has the same relative path as a file or module already on the server, the new file or module replaces the existing one. Otherwise, it is added to the deployed application.
Rollout Update	Sequentially updates an application installed on multiple cluster members across a cluster. After you update an application's files or configuration, click **Rollout Update** to install the application's updated files or configuration on all cluster members of a cluster on which the application is installed. **Rollout Update** does the following for each cluster member in sequence: 1. Saves the updated application configuration. 2. Stops all of the cluster members on one node. 3. Updates the application on the node by synchronizing the configuration. 4. Restarts the stopped cluster members. 5. Repeats steps 2 through 4 for all of the nodes that have cluster members. Use **Rollout Update** if the application is deployed on one or more clusters spread across multiple nodes. This action reduces the amount of time that any single cluster member is unavailable to serve requests to the smallest interval possible. Pending IIOP transactions will complete before a cluster member stops; in-flight HTTP and JMS transactions might be lost while the cluster member is stopping. For an application server without clusters, use **Update** and then save and synchronize the node instead. For a standalone application server, simply update and save.
Remove File	Deletes a file from the deployed application or module. This button deletes the file from the configuration repository and from the file system of all nodes where the file is installed. If the application or module is deployed on a cluster, after removing a file click **Rollout Update** to roll out the changes across the entire cluster.
Export	Opens the Export Application EAR files page so you can export an enterprise application to an EAR file. Use the **Export** action to back up a deployed application and to preserve its binding information.
Export DDL	Opens the Export Application DDL files page so you can export DDL files in the EJB modules of an enterprise application.

Table 37. Buttons for administering enterprise applications (continued)

Button	Resulting action
Export File	Accesses the Export a file from an application page, which you use to export a file of an enterprise application or module to a location of your choice. If the browser does not prompt for a location to store the file, click **File** > **Save as** and specify a location to save the file that is shown in the browser.

For more information, see Deploying and administering enterprise applications in the WebSphere Application Server Information Center.

Administering the throughput of SCA requests

For each Service Component Architecture (SCA) module deployed, requests being processed are held on queue points and in the data store for messaging engines. You can display the data for SCA requests, and take any appropriate action to manage the throughput of SCA requests.

About this task

When an SCA module is running, requests normally flow through the enterprise service bus without needing to be managed. Sometimes, you might want to check the throughput of a request, check the contents of a request, or if some problem has occurred, delete a request. You might also want to take other actions such as to monitor the overall throughput of requests, or change the reliability setting for requests.

Requests are handled as messages by the service integration technologies of the underlying WebSphere Application Server. For this reason, actions to manage requests are managed by using the WebSphere Application Server tasks to act on service integration messages.

This topic provides an overview of the main tasks that you might consider using, and links into the WebSphere Application Server tasks for more detailed information.

Procedure

- Listing messages on a message point

  SCA requests that are being processed are held on queue points of the SCA.SYSTEM.bus. You can list the SCA requests either through a queue destination for a component of the SCA module, or through the messaging engine that hosts a queue point; for example: **Service integration** > **Buses** > **SCA.SYSTEM.localhostNode01Cell.Bus** > **Destinations** > **StockQuoteService_Export** > **Queue points** > **StockQuoteService_Export@localhostNode01.server1-SCA.SYSTEM.localhostNode01Cell.Bus** > **Runtime** > **Messages**
- Deleting messages on a message point

  Under exceptional circumstances, you might need to delete one or more messages that exist on a message point for a selected bus destination or messaging engine. You should not normally need to delete messages on a message point. This task is intended as part of a troubleshooting procedure.
- Viewing data in the data store for a messaging engine.

A messaging engine maintains both volatile and durable data in its data store, including messages, transaction states, and communication channel states. You can use the database tools to view data in the data store for a messaging engine.

- Changing message reliability for a destination

  Messages have a quality of service attribute that specifies the reliability of message delivery. You can select a reliability to suit your requirements for assured delivery, and system performance.

Viewing data in a data store:

A messaging engine maintains both volatile and durable data in its data store, including messages, transaction states, and communication channel states. You can use the database tools to view data in the data store for a messaging engine.

Before you begin

Before you can use the **ij** tool to view data in an embedded Derby data store for a messaging engine, you must have stopped the messaging engine.

About this task

Volatile data is lost when a messaging engine stops, in either a controlled or an uncontrolled manner. Durable data is available after the server restarts.

In some cases, you might want to view the data in a data store; for example, to examine the messages being processed by the messaging engine.

You can use the database tools for the data store to view data in the data store for a messaging engine. For example, if the messaging engine uses the embedded Derby database, you can use the ij tool to view request messages.

Procedure

1. Start the ij tool. Windows On Windows complete the following sub-steps:
   a. Open a command window
   b. Change directory to *profile_root*`\derby\bin\embedded`
   c. Type `ij.bat`

   AIX HP-UX UNIX Linux Solaris On non-Windows platforms, complete the following sub-steps:
   a. Open a command window
   b. Change directory to *profile_root*`/derby/bin/embedded`
   c. Type `./ij.sh`
2. Open the data store for the messaging engine. Use the ij tool to complete the following sub-steps:
   a. Connect to the required database file.

      For a messaging engine, the database is stored in the directory *profile_root*`/profiles/`*profile_name*`/databases/com.ibm.ws.sib` and has the name of the messaging engine; for example, for the default standalone profile on Windows, the database file for the messaging engine localhostNode01.server1-SCA.SYSTEM.localhostNode01Cell.Bus (for server1 on the SCA.SYSTEM bus) is:

profile_root`\profiles\default\databases\com.ibm.ws.sib\localhostNode01.server1-SCA.SYSTEM.localhostNode01Cell.Bus`

   b. Use the ij tool to issue SQL commands and view data.

1) Change directory to *install_root*/derby/bin/embedded
2) Type ./ij.sh
3) Type protocol 'jdbc:derby:' ;
4) Type connect '*profile_root*/profiles/*profile_name*/databases/com.ibm.ws.sib/*database_name*' ;

c. Optional: To display more help about using ij, type help ; at the ij> prompt.

Changing message reliability for a bus destination:

Messages have a quality of service attribute that specifies the reliability of message delivery. You can select a reliability to suit your requirements for assured delivery, and system performance.

About this task

The administrator can specify the reliability setting on bus destinations, or the reliability can be specified by individual producers (typically under application control through an API call). The administrator can specify whether the default reliability for the destination can be overridden by a producer, and the maximum reliability that attached producers can request.

To browse or change the message reliability setting of a destination, use the administrative console to complete the following steps:

Procedure

1. Click **Service integration** > **Buses** in the navigation pane.
2. Click the name of the bus on which the destination exists in the content pane.
3. Click **Destinations**
4. Click the destination name. This displays the details page for the destination.
5. Review the reliability properties. The following properties control the message reliability for the destination:

   Default reliability
   The reliability assigned to a message produced to this destination when an explicit reliability has not been set by the producer.

   Maximum reliability
   The maximum reliability of messages accepted by this destination.

   These properties can have values from the following list:

   Best effort nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

   Express nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

   Reliable nonpersistent
   Messages are discarded when a messaging engine stops or fails.

   Reliable persistent
   Messages may be discarded when a messaging engine fails.

Assured persistent
Messages are not discarded.

For more information about using these properties to control message reliability, see Message reliability levels.

6. Review whether producers can override the default reliability setting.

   Enable producers to override default reliability
   Select this option to enable producers to override the default reliability that is set on the destination.

7. Optional: Change the destination properties to suit your needs.

   You can further refine the configuration of a destination by setting other properties to suit your needs, as described in Configuring bus destinations.

8. Click **OK**.
9. Save your changes to the master configuration.

Doing more with service applications and service modules

You can use the WebSphere administrative console not only to manage service modules themselves, but also the resources used by the modules and the applications that contain the modules. You also can use commands to do these tasks.

About this task

The more routine tasks for managing mediation modules are described in "Administering service modules in the administrative console" on page 413. For more advanced tasks, see the subtopics below.

Managing resources for service modules:

Service modules use resources provided by the service integration technologies of WebSphere Application Server. Service modules can also make use of a range of resources, including those provided by the Java Message Service (JMS) and Common Event Infrastructure. To administer the resources for service modules, you can use the WebSphere administrative console, commands, and scripting tools.

For more information about managing resources for service modules, see the related topics.

Note: JNDI resources must not be shared across clusters. A cross-cluster module requires that each cluster have different JNDI resources. For more information, see "Considerations for installing service applications on clusters" on page 908.

Service integration technologies

Service integration resources, such as bus destinations, enable a service module to use service integration technologies. Queue destinations are used by the SCA run time exploited by the service module as a robust infrastructure to support asynchronous interactions between components and modules. When you install a service module into WebSphere ESB, the destinations used by a module are defined on a member of the SCA.SYSTEM bus. These bus destinations are used to hold messages that are being processed for components of the service module that use asynchronous interactions:

Queue sca/*module_name*
This is the destination used to buffer asynchronous requests sent to module *module_name*

Queue sca/*module_name***/exportlink/***export_name*
This is the destination used by the export to send asynchronous requests to the module. Requests are routed to the component target linked to the export.

Queue sca/*module_name***/importlink/***import_name*
This is the destination used by the import to send asynchronous requests out of the module. Requests are routed to the module export linked to the import.

Queue sca/*module_name***/import/sca/dynamic/import/scaimport [for SCA binding]**

Queue sca/*module_name***/import/sca/dynamic/import/wsimport [for Web service binding]**

Queue sca/contextStore/*module_name*

For each of the destinations, a queue point is also created and defined on the messaging engine of the relevant bus member.

You can deploy and use service modules without needing to manage these resources. However, you might want to adjust the configuration of the resources (for example, to modify the maximum messaging quality of service used) or to use them in locating messages for troubleshooting.

Java Message Service (JMS)

JMS resources enable a service module to use asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. The JMS support used depends on the JMS binding of the module. For example, a module with a JMS binding uses a JMS connection factory configured on the default messaging provider provided by the underlying WebSphere Application Server, while a module with a WebSphere MQ JMS binding uses a JMS connection factory configured on WebSphere MQ as the JMS provider. To manage use of the Java Message Service, you can administer the following resources:

JMS connection factory
A JMS connection factory is used to create connections to the associated JMS provider of JMS destinations, for both point-to-point and publish/subscribe messaging. Use connection factory administrative objects to manage JMS connection factories for the provider.

JMS queue
A JMS queue is used as a destination for point-to-point messaging. Use JMS queue destination administrative objects to manage JMS queues for the provider.

JMS topic
A JMS topic is used as a destination for publish/subscribe messaging. Use topic destination administrative objects to manage JMS topics for the provider.

JMS activation specification
A JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

JMS listener port
A JMS listener port defines the association between a connection factory, a destination, and a message-driven bean. This enables deployed message-driven beans associated with the port to retrieve messages from the destination.

Common Event Infrastructure (CEI)

CEI resources enable a service module to use standard formats and mechanisms for managing event data. To manage use of the Common Event Infrastructure, you can administer the following resources:

Data Store Profile
This profile defines properties used by the default data store. The default data store is the data store supplied by the Common Event Infrastructure.

Emitter Factory Profile
This profile defines the options for an event emitter.

Event Bus Transmission Profile
This profile defines the EJB entry into the event bus.

Event Group Profile
This profile defines a list of events which are determined through selector expressions. JMS queues and a JMS topic can be associated with each event group. If the event server distribution service is enabled and an event matches an event group the event is distributed to any topic or queues configured for that particular event group.

Event Server Profile
This profile defines the properties for the event server.

Filter Factory Profile
This profile defines the properties of a filter. The filter uses the filter configuration string to determine whether an event will be passed to the bus.

JMS Transmission Profile
This profile defines a JMS queue entry into the event bus. It defines the JNDI names for a JMS queue and queue connection factory.

Managing service integration in applications:

This set of topics provides information about the service integration technologies. Service integration is implemented as a group of messaging engines running in application servers (usually one engine to one server) in a cell.

A service integration bus is a form of managed communication that supports service integration through synchronous and asynchronous messaging. A bus

consists of interconnecting messaging engines that manage bus resources. The members of a service integration bus are the application servers and clusters on which the messaging engines are defined.

Service Integration Bus Browser:

The Service Integration Bus Browser provides a single location for browsing and performing day-to-day operational tasks on service integration buses.

Examples of day-to-day operations include browsing service integration buses, viewing runtime properties for messaging engines, or managing messages on message points. The browser is not intended as a bus configuration tool.

When you access the Service Integration Bus Browser by clicking **Service Integration** > **Service Integration Bus Browser**, two panes open to the right of the standard console navigation pane:

Navigation tree pane
: This pane contains a navigation tree that allows you to browse the service integration buses configured on the system.

Content pane
: This pane contains collection and detail pages for the buses and their individual components, such as messaging engines, queue points, destinations, publication points, and mediation points.

  Note that not all pages can be edited when accessed from a link in the navigation tree. See the online help for the browser for more detail, including how to access versions of the page that can be edited.

When you click an item in the navigation tree pane, its corresponding collection or detail page opens in the content pane.

Table 38 lists and describes the icons associated with each item in the navigation tree.

Table 38. Icons in the Service Integration Bus Browser

Icon	Description
	Indicates a collection of buses, destinations, queue points, publication points, or mediation points, depending on where in the navigation tree it appears.
	Indicates a service integration bus.
	Indicates a messaging engine.
	Indicates a service integration bus member.

Related tasks:

"Viewing the service integration bus" on page 1110
Use the Service Integration Bus browser on the administrative console to view the service integration bus.

Working with targets:

Targets provide additional flexibility by allowing you to modify processing by changing the target configured for a reference.

A component can call a component in another module to minimize the time and cost of building an application. Targets provide additional flexibility: to allow your

installed applications to benefit from advances in processing or other changes, you can use the administrative console to change the endpoint of a cross-module invocation, without rewriting or redeploying the application.

To take advantage of the flexibility provided, you must understand how the system names the targets. The link from the calling module must connect to the correct target.

Target names

Target names are derived from how the calling component invokes the target. The names have the following format:

Invocation type
: **Name format**

Synchronous
: A name that follows the Java Naming and Directory Interface (JNDI) format, for example:

  `folder/export/fullpath_to_target/target_component_name`

Asynchronous
: A name with the format

  `folder/calling_component_name/ full_path_to_target_component/target_component_name`

Multiple destinations
: This name is the same as an asynchronous invocation but the target sends a message to multiple destination components.

Related tasks:

"Changing import targets"
Changing the target of a reference provides applications with the flexibility of taking advantage of advances in components as they happen without recompiling and reinstalling the application.

Changing import targets:

Changing the target of a reference provides applications with the flexibility of taking advantage of advances in components as they happen without recompiling and reinstalling the application.

Before you begin

Before changing the target for a reference you must:

- Make sure the new target uses the same data object type
- Know whether the module is synchronously or asynchronously invoking the target
- Know whether the reference targets a single or multiple services

About this task

Change the target of an import from a module when another service with the same interface as the original target provides new or improved functionality that your module can use.

Procedure

1. Stop the module that contains the reference that you are changing.
   a. Using the administrative console, display the Service Component Architecture (SCA) modules.

      Navigate to this panel using **Applications > SCA Modules**
   b. Select your module and press **Stop**. The display updates to show the application as stopped.
2. Change the target destination of the reference.

   How you make the change depends on how the module invokes the target.

Type of invocation	How to change
Single target service	1. Using the administrative console, display the SCA Modules. Navigate to the panel using **Applications > SCA Modules**. 2. From the displayed list, select the module that contains the import that references the target to change. 3. Expand the list of imports by clicking the plus sign (+) next to **Imports**. 4. Select the import to change from the list. 5. In the **Target** area, select the **Module** from the list. 6. After the **Export** list refreshes, select the export for the new target. 7. Save the change by clicking **OK**.
Multiple target services	1. Display the buses on the system on which the module resides. Navigate to the panel using **Service Integration > Buses**. 2. Select the **SCA.System.*cellname*.Bus** 3. Display the destination targets for the bus by clicking **Destinations**. 4. Select the destination that represents the import that connects the calling module to the targets. This identifier will contain the word import. 5. Display the list of properties by clicking **Context properties**. 6. Select the property to change by clicking on the targets property in the list. 7. Change the **Context value** field to the new destination targets. 8. Return to the **Context properties** panel by clicking **OK**. 9. Save the change by clicking **OK**.

3. Save your changes. Click **Save** when prompted.

What to do next

Start the module and make sure the module receives the expected results.

Using commands to manage service applications:

You can manage service applications using commands. The commands can be used within scripts.

Before you begin

Use the wsadmin tool to run service application commands.

About this task

You can use the wsadmin tool in different ways. You can use the tool interactively, as an individual command, or with a script. Running multiple commands in a script is useful if you are administering multiple machines.

WebSphere ESB includes commands that display SCA modules and their imports and exports and that change the details of import and export bindings.

Managing service modules using commands:

You can use a command to list the service modules that have been deployed to WebSphere ESB. You can also view information about a service module and make changes to properties associated with the service module.

Before you begin

Use the wsadmin tool to run WebSphere ESB commands.

About this task

You can run commands individually or in a script. Running multiple commands in a script is useful if you are administering multiple hosts or producing regular reports.

Listing service modules using commands:

You can use a command to list the service modules that have been deployed to WebSphere ESB.

About this task

Use the wsadmin tool to list service modules.

Because mediation modules are types of SCA modules, you use the listSCAModules command to display the deployed mediation modules.

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

List the deployed SCA modules.

```
AdminTask.listSCAModules()
```

Results

Lists the SCA modules that have been deployed to WebSphere ESB, and the applications they are associated with. The output is returned in the format:

module name:application name

This format makes it easier for scripts to parse the output and extract names for use in subsequent commands.

Displaying the properties of a service module using commands:

You can use a command to show the properties for a specified service module.

About this task

Use the wsadmin tool to show the properties of a particular service module. To display the properties, you need to know the module name.

Note that you can use these commands individually, or you can use commands in combination. For example, you can parse the information returned by the first command (such as the module name) to determine the value to use in the second command.

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

1. Optional: List the SCA modules.
   `AdminTask.listSCAModules()`
2. Display the properties of a particular SCA module.
   `AdminTask.showSCAModuleProperties('-moduleName moduleName')`

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

Results

The properties of the specified SCA module are displayed.

The output of the listSCAModules command is in this format:

```
myModule:myModuleApp: :myModule: :\nyourModule_v1_0_0_yourCellId:
yourModule_v1_0_0_yourCellIdApp:6.0.0:yourModule:yourCellId:
```

The output of the showSCAModuleProperties command is in this format:

```
[myGroup]myProperty1=myValue1\n[myGroup]myProperty2=myValue2
```

Changing a service module property using commands:

You can use a wsadmin scripting command to change a property value for a specified service module.

About this task

Use the wsadmin tool to change property values. To change the properties, you need to know the module name.

Note that you can use these commands individually, or you can use commands in combination. For example, you can parse the information returned by the first command (such as the module name) to determine the value to use in the second command.

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

1. Optional: List the deployed SCA modules by using the following wsadmin scripting command: .

   ```
   AdminTask.listSCAModules()
   ```

2. Optional: List the properties for a particular SCA module by using the following wsadmin scripting command:

   ```
   AdminTask.showSCAModuleProperties('-moduleName moduleName')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

3. Modify a module property for the SCA module by using the following wsadmin scripting command:

   ```
   AdminTask.modifySCAModuleProperty('-moduleName moduleName
   -propertyName propertyName -newPropertyValue newpropertyValue')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

Results

The property value for the specified SCA module property is changed.

Working with imports:

You can use a command to list the imports of any service module deployed to WebSphere ESB. You can also use commands to list the details of an import or the bindings associated with the import and to change the settings of import bindings.

Displaying imports and associated details using a command:

You can use a wsadmin scripting command to list the imports associated with a service module deployed to WebSphere ESB. You can also display details of a particular import.

About this task

Use the wsadmin tool to list the imports associated with a service module and to display the details of a service module import.

To show the details of a particular service module import, you need to know the module name and the import name.

Note that you can use these commands individually, or you can use commands in combination. For example, you can parse the information returned by the first command (such as the module name) to determine the value to use in the second command.

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

1. Optional: List the deployed SCA modules by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAModules()
   ```

2. List the imports for a particular SCA module by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAImports('-moduleName moduleName')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

3. Show the details of a particular SCA module import by using the following wsadmin scripting command:

   ```
   AdminTask.showSCAImport('-moduleName moduleName
   -import importName')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

Results

Displays the imports associated with a module and the details for a particular SCA module import.

Displaying an import binding using a command:

You can use a wsadmin scripting command to display the import bindings of a service module deployed to WebSphere ESB.

About this task

Use the wsadmin tool to display the bindings of a particular service module import.

Note: It is possible for an SCA module not to have any imports.

To show the bindings of a particular service module import, you need to know the module name and the import name.

You can also display information about a specific type of import binding, such as Web service or JMS.

Note that you can use these commands individually, or you can use commands in combination. For example, you can parse the information returned by the first command (such as the module name) to determine the value to use in the second command.

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

1. Optional: List the deployed SCA modules by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAModules()
   ```

2. Optional: List the imports for a particular SCA module by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAImports('[-moduleName moduleName]')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

3. Show the import binding for a particular import by using the following wsadmin scripting command:

   ```
   AdminTask.showSCAImportBinding('[-moduleName moduleName
   -import importName]')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

Results

Displays the import binding for a particular SCA module import.

The output of the showSCAImportBinding command depends upon the type of binding. For example, for an adapter (EIS) import binding, the output would be in the following format:

```
importBinding:type=AdapterImportBinding
```

Changing an import binding using commands:

You can use a wsadmin scripting command to change the import bindings of service modules deployed to WebSphere ESB.

About this task

Use the wsadmin tool to change import binding properties associated with a service module.

Note: It is possible for an SCA module not to have any imports.

To change the properties, you need to know the module name.

Note that you can use these commands individually, or you can use commands in combination. For example, you can parse the information returned by the first command (such as the module name) to determine the value to use in the second command.

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

1. Optional: List the deployed SCA modules by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAModules()
   ```

2. Optional: List the imports for a particular SCA module by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAImports('[-moduleName moduleName]')
   ```

   Note: In addition to specifying the *moduleName*, you have the option of specifying the *applicationName*. Providing an *applicationName* improves performance.

3. Modify an import binding using the modifySCAimport*bindingType*Binding command, where *bindingType* is the actual type of binding, as in the following list:

   - **modifySCAImportEJBBinding**
   - **modifySCAImportHttpBinding**
   - **modifySCAImportJMSBinding**

     Note: The **modifySCAImportJMSBinding** command is used for the JMS binding, the generic JMS binding, and the WebSphere MQ JMS binding.
   - **modifySCAImportMQBinding**
   - **modifySCAImportSCABinding**
   - **modifySCAImportWSBinding**

   Each of these commands has its own set of parameters. For example, the **modifySCAImportSCABinding** has the following parameters:

   ```
   AdminTask.modifySCAImport('[-moduleName moduleName
   -import importName -targetModule targetModuleName
   -targetExport  targetExportName]')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

Results

Changes the import binding for a particular import.

When using the modifySCAImportSCABinding command to change the SCA export referred to by an SCA import binding, WebSphere ESB issues a warning for each import interface that is not satisfied by an export interface. WebSphere ESB compares the WSDL port type names of the import and export. If they are not the same, a warning is issued. However, if the port type names do match, WebSphere ESB assumes that the operations provided are equivalent and no warning is issued.

Example

To change an EJB import binding:

```
AdminTask.modifySCAImportEJBBinding('[-moduleName myModule
-import myImport -jndiName newjndiName
-applicationName myApplication]')
```

To change a Web service import binding:

```
AdminTask.modifySCAImportWSBinding('[-moduleName myModule
-applicationName myApplication -import myImport
-endpoint http://myTargetEndpoint]')
```

Working with exports:

You can use a command to list the exports of any service module deployed to WebSphere ESB. You can also use commands to list the details of an export or the bindings associated with the export and to change the settings of HTTP, JMS, and WebSphere MQ export bindings.

Displaying exports and associated details using a command:

You can use a wsadmin scripting command to list the exports associated with a service module deployed to WebSphere ESB. You can also display details of a particular export.

About this task

Use the wsadmin tool to list the exports associated with a service module and to display the details of a service module export.

Note: It is possible for an SCA module not to have any exports.

To show the details of a particular service module export, you need to know the module name and the export name.

Note that you can use these commands individually, or you can use commands in combination. For example, you can parse the information returned by the first command (such as the module name) to determine the value to use in the second command.

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

1. Optional: List the deployed SCA modules by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAModules()
   ```

2. List the exports for a particular SCA module by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAExports('[-moduleName moduleName]')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

3. Show the details of a particular SCA module export by using the following wsadmin scripting command:

   ```
   AdminTask.showSCAExport('[-moduleName moduleName
   -export exportName]')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

Results

Displays the exports associated with a module and the details for a particular SCA module export.

The output of the showSCAExport command is in this format:

```
export:name=exportName,description=null
interface:type=type,portType=portType
```

Displaying an export binding using a command:

You can use a wsadmin scripting command to display the export bindings of a service module deployed to WebSphere ESB.

About this task

Use the wsadmin tool to display the bindings of a service module export.

Note: It is possible for an SCA module not to have any exports.

To show the bindings of a particular service module export, you need to know the module name and the export name.

You can also display information about a specific type of export binding, such as Web service or JMS.

Note that you can use these commands individually, or you can use commands in combination. For example, you can parse the information returned by the first command (such as the module name) to determine the value to use in the second command.

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

1. Optional: List the deployed SCA modules by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAModules()
   ```

2. Optional: List the exports for a particular SCA module by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAExports('[-moduleName moduleName]')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

3. Show the export binding for a particular export by using the following wsadmin scripting command:

   ```
   AdminTask.showSCAExportBinding('[-moduleName moduleName
   -export exportName]')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

Results

Shows the export binding for a particular SCA module export. The information displayed depends upon the type of binding. If an export has no binding specified, the run time assumes that the binding is of type SCA.

The output of the showSCAExportBinding command depends upon the type of binding. For example, for Web service export bindings, the output is in the following format:

```
exportBinding:type=WebServiceExportBinding,port=_:portType,service=_:serviceName
```

Changing an export binding using commands:

You can use a wsadmin scripting command to change the export bindings of service modules deployed to WebSphere ESB. You can change the properties of HTTP, JMS, and WebSphere MQ export bindings.

About this task

Use the wsadmin tool to change export binding properties associated with a service module.

Note: It is possible for an SCA module not to have any exports.

To change the bindings of a particular service module export, you need to know the module name and the export name.

Note that you can use these commands individually, or you can use commands in combination. For example, you can parse the information returned by the first command (such as the module name) to determine the value to use in the second command.

You can change the following types of export bindings:

- HTTP
- JMS (which applies to generic JMS and MQ JMS as well)
- MQ

Note: The following procedure uses Jython syntax. See the individual command descriptions for information about using Jacl.

Procedure

1. Optional: List the deployed SCA modules by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAModules()
   ```

2. Optional: List the exports for a particular SCA module by using the following wsadmin scripting command:

   ```
   AdminTask.listSCAExports('[-moduleName moduleName]')
   ```

   Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

3. Modify an export binding using the modifySCAexport*bindingType*Binding command, where *bindingType* is the actual type of binding, as in the following list:

- **modifySCAExportHttpBinding**
- **modifySCAExportJMSBinding**
- **modifySCAExportMQBinding**

Each of these commands has its own set of parameters. For example, the **modifySCAExportJMSBinding** has the following parameters:

```
AdminTask.modifySCAExportJMSBinding
('[-moduleName moduleName -export exportName
-type JMS -sendDestination  sendDestinationName]')
```

Note: In addition to specifying the module name, you have the option of specifying the application name and other optional parameters. Providing an application name improves performance.

Results

Changes the export binding for a particular export.

Example

To change an MQ export binding:

```
AdminTask.modifySCAExportMQBinding('[-moduleName myModule
-export exportName -sendDestination sendDestinationName]')
```

Administering access to WSRR

You can create, configure, and view all your WebSphere Service Registry and Repository (WSRR) access definitions using the administrative console.

Before you begin

Install and configure WSRR as described in the WSRR Information Center. For more information, see: Installing WebSphere Service Registry and Repository.

About this task

WebSphere ESB supports the use of the WSRR product, which allows the storage and retrieval of services endpoints and mediation policies.

WSRR definitions, and their connection properties, are the mechanism used to connect to a registry instance and look up a Web service or a mediation policy. WSRR is installed as an enterprise application and provides a Web service interface that allows you to connect Endpoint Lookup mediation primitives and Policy Resolution mediation primitives to a registry instance (as part of a mediation flow). You can use the Endpoint Lookup mediation primitive to look up services, and the Policy Resolution mediation primitive to find mediation policies. To display a list of all your WSRR definitions, use the administrative console to complete the following step.

Procedure

In the navigation pane, expand **Service integration** > **WSRR definitions**.

Results

The content pane displays a list of all the WSRR definitions. Each definition holds a set of properties that identify a WSRR instance, and specify how you can access the WSRR instance.

What to do next

Click on a WSRR definition to display the general properties for that definition. You can access the detailed connection properties from the content pane.

Related concepts:

WebSphere Service Registry and Repository
The WebSphere Service Registry and Repository (WSRR) product allows you to store, access, and manage information about service endpoints and mediation policies. You can use WSRR to make your service applications more dynamic, and more adaptable to changing business conditions.

Related reference:

WSRR administration commands

Creating a new WSRR definition

You can create a new WSRR definition using the administrative console.

About this task

Each definition holds a set of properties that identify a WSRR instance and specify how you can access it. To create a new WSRR definition, use the administrative console to perform the following steps.

Procedure

1. Expand **Service integration** > **WSRR definitions**, in the navigation pane. The WSRR definitions page is displayed in the content pane, which shows a list of all the WSRR definitions.
2. Click **New**. The WSRR definitions configuration page displays.
3. Complete the following property fields:

   WSRR definition name
   This is required and must be unique inside the cell. It is the administrative name for this WSRR definition.

   Description
   You have the option to enter a description for the definition.

   Default WSRR definition
   This indicates whether this definition is the default. If this is the first definition you have created, it will automatically be set as the default. You cannot update the default from this page.

   Timeout of cache
   The time (in seconds) after which the results in the WSRR cache expire and will be refreshed. This can be changed from the default value. If you specify a value of 0 (zero), then results are never cached. The cache is used for storing information about service endpoints and mediation policies, retrieved from the registry.

   Connection type
   Your only choice is **Web service**.
4. Click **Apply** to save these properties.

5. Click **Connection properties**, under **Additional properties** in the content pane. The Connection properties configuration page displays.
6. Complete the following property fields:

   Connection type
   This is set when the registry definition is created, and cannot be changed.

   Registry URL
   The URL to be used to connect to the WSRR instance. The default value is: `http://localhost:9080/WSRRCoreSDO/services/WSRRCoreSDOPort`

   If you want to retrieve endpoints from WSDL that has SOAP 1.2 bindings, you must use the Version 6.2 Web service interface URL, which is generally: `http://<server>:<port>/WSRR6_2/services/WSRRCoreSDOPort`.

   Note: If you call the Version 6.2 URL from a client generated for a previous version of WSRR, the client might fail if it cannot correctly process the data that is returned.

   Authentication alias
   The alias to be used to authenticate with the WSRR instance.

   SSL Configuration
   The Secure Sockets Layer (SSL) configuration to be used to communicate with a secured WSRR instance.
7. Click **Apply** to save these properties.
8. Click **Test Connection** to test the connection to the WSRR instance that has been defined. If successful the following message is displayed:

   `The test connection operation for WSRR definition` *`myWSRRDef`* `was successful`
9. Click **Save** to apply your changes to the master configuration.
10. Click **Save** again to complete the procedure.

Results

A new WSRR definition is created.

Related tasks:

"Setting a default WSRR definition" on page 495
You can set a WSRR definition to be the default definition, from the administrative console.

Deleting a WSRR definition

You can delete a WSRR definition from the administrative console.

About this task

To delete a WSRR definition, use the administrative console to perform the following steps.

Note: You cannot delete the default WSRR definition unless it is the only one. If you want to delete the current default definition, you must first set a different definition to be the default.

Procedure

1. Expand **Service integration** > **WSRR definitions** in the navigation pane. The WSRR definitions page displays in the content pane, showing a list of all the WSRR definitions.
2. Select the check box next to the WSRR definition you want to delete.
3. Click **Delete**. The definition disappears from the collection panel.
4. Click **Save** to apply your changes to the master configuration.

Results

The WSRR definition is deleted.

Related tasks:

"Setting a default WSRR definition"
You can set a WSRR definition to be the default definition, from the administrative console.

Setting a default WSRR definition

You can set a WSRR definition to be the default definition, from the administrative console.

About this task

To set a WSRR definition to be the default, use the administrative console to perform the following steps.

Procedure

1. Expand **Service integration** > **WSRR definitions** in the navigation pane. The WSRR definitions page displays in the content pane, showing a list of all the WSRR definitions.
2. Select the check box next to the WSRR definition you want to set as the default.
3. Click **Set as default**. A **Yes** displays in the Default column for the selected WSRR definition. If another WSRR definition was previously set to be the default, a **No** displays in the Default column for that WSRR definition.
4. Click **Save** to apply your changes to the master configuration.
5. Click **Save**.

Results

The selected WSRR definition is set as the default definition.

Note: You can sort the Default column, so that the default definition appears at the top, or the bottom, of the list.

Related tasks:

"Deleting a WSRR definition" on page 494
You can delete a WSRR definition from the administrative console.

"Creating a new WSRR definition" on page 493
You can create a new WSRR definition using the administrative console.

Clearing the cache associated with a WSRR definition

You can clear the cache associated with a WSRR definition, using the administrative console.

About this task

When mediation flows query a registry, the information returned is stored in a cache associated with the WSRR definition (information relating to service endpoints and mediation policies can be cached).

Updates to WSRR might invalidate query results that have been cached. Therefore, you can clear cached results, so that the next registry query uses the new values.

Procedure

1. Expand **Service integration** > **WSRR definitions** in the navigation pane. The WSRR definitions page displays in the content pane, showing a list of all the WSRR definitions.
2. Select the check boxes next to the WSRR definitions whose caches you want to clear.
3. Click **Clear all active caches**.

Results

The caches, associated with the WSRR definitions you selected, are cleared.

What to do next

If you want to clear the cache using commands, see: *Example: Clearing the cache using commands*.

Example: Clearing the cache from the command line:

This example shows you how to clear the cache associated with a WSRR definition, from the command line.

Before you begin

This example uses the command-line scripting tool, wsadmin.

About this task

When mediation flows query a registry, the information returned is stored in a cache associated with the WSRR definition (information relating to service endpoints and mediation policies can be cached).

Before you can clear the cache, using wsadmin, you need to have a reference to an MBean of type ServiceRegistry. You must get a ServiceRegistry MBean for each instance of the registry you need to clear; then you use the MBeans to clear the cache.

Note: A ServiceRegistry MBean is created when an WSRR instance, as defined by a WSRR definition, is queried for the first time by an Endpoint Lookup mediation primitive or a Policy Resolution mediation primitive.

Procedure

1. From a command line, start wsadmin.
2. From wsadmin, retrieve a reference to a specific ServiceRegistry MBean. For example,

```
wsadmin>set wsrrDef [$AdminControl completeObjectName
WebSphere:type=ServiceRegistry,mbeanIdentifier=testWSRR1,*]
```

This example, uses JACL to set a reference to a ServiceRegistry MBean in the variable *wsrrDef*.

Note: In a network deployment environment there might be several instances of each ServiceRegistry MBean, each of which would require the cache to be cleared.

3. Invoke the **expireCache** operation on the ServiceRegistry MBean whose reference you have retrieved. For example:

```
wsadmin>$AdminControl invoke $wsrrDef expireCache
```

.

Results

The cache, associated with an instance of a ServiceRegistry MBean, is cleared.

Administering relationships

The relationship manager is a tool for manually controlling and manipulating relationship data to correct errors found in automated relationship management or provide more complete relationship information. In particular, it provides a facility for retrieving as well as modifying relationship instance data.

How the relationship manager works

The relationship manager allows you to configure, query, view, and perform operations on relationship runtime data, including roles and their data. You create relationship definitions with the relationship editor. At run time, instances of the relationships are populated with the data that associates information from different applications. This relationship instance data is created when the maps or other WebSphere ESB components run and need a relationship instance. The relationship service exposes a set of application programming interfaces (API's) to retrieve relationship metadata and to create, retrieve, and manipulate the instance data. The data is stored in the relationship tables that are specified in the relationship definition. The relationship manager provides a graphical user interface to interact with the relationships and relationship instances.

For each relationship instance, the relationship manager can display a hierarchical listing of its roles. Each role in the relationship has instance data, properties, and key attributes. The relationship tree also provides detailed information about each of the roles in the relationship instance, such as the type of entity, its value, and the date it was last modified. A relationship instance ID is automatically generated when the relationship instance is saved in the relationship table. The relationship manager displays this instance ID at the top level of the relationship tree.

Uses of the relationship manager

You can use the relationship manager to manage entities at all levels: the relationship instance, role instance, and attribute data and property data levels. For example, you can use the relationship manager to:

- Browse and inspect the values for existing relationships
- Create and delete relationship instances

- Modify the contents of a relationship instance, such as adding and deleting role instances
- Edit the data of a relationship role instance like role properties and logical state
- Activate and deactivate role instances
- Get role instances, given the key attribute, start and end date, and property value
- Export an existing static relationship instance (from one platform) to an RI or CSV file, then use the relationship manager to import the RI or CSV file into a new environment (a different platform from the first)
- Salvage a situation when problems arise. For example, when corrupt or inconsistent data from a source application has been sent to the generic and destination application relationship table, you can use the relationship manager to rollback the data to a point in time when you know the data is reliable

For more information on relationships, see the IBM Integration Designer Information Center and the topics on the relationship service in the WebSphere ESB Information Center.

Viewing relationships

You can view a list of relationships in the system, including the relationship name, display name, and static and identity attributes.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as a monitor, an operator, a configurator, or an administrator.

About this task

To view the list of relationships in the system, do the following:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.

   The information is displayed in table format. Each relationship type is a link.

   Tip: You can customize the number of rows to display at one time. Click **Preferences** and modify the **Maximum row** field value. The default is 25.

Viewing relationship details

You can view detailed information for the selected relationship, including the relationship name, display name, associated roles with their attributes, property values, and static and identity attributes.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as a monitor, an operator, a configurator, or an administrator.

About this task

To view detailed information for the selected relationship, complete the steps in the Procedure section:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. You can view the relationship details in two ways:
   a. Click the relationship name.
   b. Select the radio button next to the relationship name and click **Details**.

   The relationship details include role attributes, which are displayed in table format and include the display name, object name, and managed attribute setting for the role.

   To return to the list of relationships, click **Relationships** from the path at the top of the page or click **Back**.

Viewing role details

You can view detailed information for the selected role, including the relationship name, role name, display name, property values, keys, role object type, and managed attribute setting.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as a monitor, an operator, a configurator, or an administrator.

About this task

To view detailed information for the selected role, complete the steps in the Procedure section:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the Relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Click a relationship name to open the Relationship Detail page.
5. Under **Role schema information**, click an associated role name to open the Role Detail page.

What to do next

To return to the Relationship Detail page, click **Relationship Detail** from the path at the top of the page or click **Back**.

Querying relationships

Use this task to make relationship-based instance queries.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as a monitor, an operator, a configurator, or an administrator.

About this task

Select a query option (**All**, **By ID**, **By property**, or **By role**) to get all or a subset of the instance data for a relationship. The return is the result set of that query and is displayed in table format with each row representing one relationship instance.

To query relationships, do the following:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs and specify the search criteria.

Option	Description
All tab	Get a list of all instances in the relationship. You can select to display all activated, all inactivated, or all activated and inactivated relationship instance data.
By ID tab	Get relationship instances in the range of the starting and ending instance identifiers. If you leave one field blank, the query returns only the single instance. The query returns all of the roles for the instances it finds.
By property tab	Get relationship instances by specific property values.
By role tab	Get relationship instances based on a role name, key attribute value, date range during which the role was created or modified, or specific property value.

6. After you have specified the query parameters, you have the following options:
   - Click **OK** to display the result data from the query.
   - Click **Cancel** to discard any changes made and return to the list of relationships.

Querying relationship data using database views:

You can use views in a database to query relationship data without using the relationship manager.

You can use your database views to directly query relationship data stored on the database. When you create a new relationship database table, a corresponding SQL

view is automatically created. These views are essentially encapsulations of the relationship data stored in database tables. You can use these views to populate, query relationship data, or both by:

- using SQL statements with a DB client (for example, with the DB2 command center)
- using JDBC to run SQL statements with a Java program

In either case, you can use the SQL views in the same manner as you would for tables. You can use this technique as an alternative method to the Relationship Manager application to directly populate large sets of application-specific data by using SQL statements into your relationship database(s). You can also use this technique to import data from a flat-text file into a database table

Relationship database SQL views are created based on data contained in tables located elsewhere in the data source. The view will exist even when the database table itself is empty. Each view is has its own unique name which follows this convention: "V_"+*relationship_display_name*+"_"*role_display_name*+"_"+*uuid* (notice that the variables are concatenated using an underscore character "_"). Both display names are limited to 20 alphanumeric characters, while the uuid is a number generated from the combination of both display names. Consequently, each view name should be unique within a data source. An example of this naming convention can be shown using these variables:

- *relationship_display_name* = SAMPLECUSTID
- *role_display_name* = MYCUSTOMER
- *uuid* = 80C (this number is generated automatically by the server)

The resulting view name would be "V_SAMPLECUSTID_MYCUSTOMER_80C". For a given relationship, you should have two corresponding views containing the same relationship display name but different role display names and uuids.

Note: For Oracle databases, the naming convention differs in this regard: only the first ten characters of the *relationship_display_name* and *role_display_name* are used.

Each view will contain the columns (including the associated properties of type, value, and nullable) listed in the following table:

Table 39. Relationship database view columns

Name	Data type	Value	Nullable?
`INSTANCEID`	Integer	The ID number used to correlate instance data between different applications.	No
ROLE_ATTRIBUTE_COLUMNS • Dynamic relationship - defined in business object • Static relationship - `DATA`	• Dynamic relationship - defined in business object • Static relationship - Varchar	The column name and type depends on the role definition. Column names are based on the key attribute names, while column types are database data types that are mapped based on key attribute type defined in role definition.	No

Table 39. Relationship database view columns (continued)

Name	Data type	Value	Nullable?
STATUS	Integer	0-4 0 – created 1 – updated 2 – deleted 3 – activated 4 – deactivated **Note:** When populating instances through views, ensure that the value for this column is 0.	Yes
LOGICAL_STATE	Integer	• 0 = activated • 1 = deactivated Ensure that you set the proper value when you populate the database with data.	No
LOGICAL_STATE_TIMESTAMP	Timestamp	Date and time when the logical state column data was last updated.	Yes
CREATE_TIMESTAMP	Timestamp	Date and time when the role instance was created.	Yes
UPDATE_TIMESTAMP	Timestamp	Date and time when the role instance was last updated.	Yes
ROLEID	Integer	ID number used to identify a role instance	No

Example

This example presented here is an identity relationship that includes three sets of data from three enterprise applications:

- Clarify
- SAP
- Siebel

The data is correlated using the WebSphere ESB relationship service. Each application contains similar customer information, with an identity relationship to correlate the information between each application.

The following three tables show the data as it is stored within each database:

Table 40. Clarify customer

Given Name	Last Name	Home Phone	ID
Jessica	Reed	111 111 11111	clarify_1
Tara	McLean	333 333 33333	clarify_2

Table 41. SAP customer

First Name	Last Name	Home Phone	ID
Jessica	Reed	111 111 11111	sap_10
Tara	McLean	333 333 33333	sap_8

Table 42. Siebel customer

Full Name	Home Phone	ID
Jessica Reed	111 111 11111	siebel_6
Tara McLean	333 333 33333	siebel_8

The customer business object definition names and elements (created in Integration Designer for each database) are shown in the following table:

Table 43. Business object definitions for customer on each database

ClarifyCustomer		SapCustomer		SiebelCustomer	
Element	Type	Element	Type	Element	Type
`givenName`	`string`	`firstName`	`string`	`fullName`	`string`
`lastName`	`string`	`lastName`	`string`		
`homePhone`	`string`	`homePhone`	`string`	`homePhone`	`string`
`clarifyId`	`string`	`sapId`	`string`	`siebelId`	`string`

An identity relationship is defined to correlate the customer information between each database. This relationship, called `ID` in this example, uses the business object elements `clarifyId`, `sapId`, and `siebelId`. These elements are used because they contain the ID data for each database, and that data is unique for each customer. The following table describes the roles that are used to correlate different databases in the relationship to a common ID used by WebSphere ESB:

Table 44. ID relationship definition

Relationship name	Role name	Business object name	Key
`ID`	`GenCustomer`	`GenCustomer`	`genId`
	`ClarifyCustomer`	`ClarifyCustomer`	`clarifyId`
	`SapCustomer`	`SapCustomer`	`sapId`
	`SiebelCustomer`	`SiebelCustomer`	`siebelId`

The full relationship name is `http://CustomerModule/ID`. The full role names are

- `http://CustomerModule/ClarifyCustomer`
- `http://CustomerModule/SapCustomer`
- `http://CustomerModule/SiebelCustomer`

You can correlate the data within the business objects contained in all three databases by using the defined relationship. The customer ID data from each database is correlated with the customer data from the other databases by sharing instance IDs. For example, Tara McLean is identified by `clarify_3 ID` in Clarify, `sap_8` in SAP, and `siebel_8` in Siebel. A unique ID is generated by the WebSphere ESB relationship service.

Tip: You can use the views to browse the relationship table content.

You can define multiple relationship instances by using the views created in the Common database. The mapping of the view name (using the naming convention as previously described) to its corresponding relationship role is captured in the `RELN_VIEW_META_T` table in the Common database. The following table shows an example of the view names for the `ClarifyCustomer`, `SapCustomer`, and `SiebelCustomer` roles:

Table 45. RELN_VIEW_META_T table

VIEW_NAME	RELATIONSHIP_NAME	ROLE_NAME
`V_ID_CLARIFYCUSTOMER_098`	`http://CustomerModule/ID`	`http://CustomerModule/ClarifyCustomer`
`V_ID_SAPCUSTOMER_515`	`http://CustomerModule/ID`	`http://CustomerModule/SapCustomer`
`V_ID_SIEBELCUSTOMER_411`	`http://CustomerModule/ID`	`http://CustomerModule/SiebelCustomer`
`V_USASTATE_ABBREVIATION_DE8`	`http://CustomerModule/USASTATE`	`http://CustomerModule/Abbreviation`
`V_USASTATE_CODE_B32`	`http://CustomerModule/USASTATE`	`http://CustomerModule/Code`
`V_USASTATE_NAME_933`	`http://CustomerModule/USASTATE`	`http://CustomerModule/FullName`

The view column definition as described in table 1 will have a `ROLE_ATTRIBUTE_COLUMN` with the following properties:

Table 46. View column definition

Column Name	Data Type	Value	Description
`KEY_ATTRIBUTE_NAME`	depends on the key attribute type	Not null	This is where the role instance data is stored. For identity relationships, the column is named by the name of the key attribute. For example, `SAPCUSTOMER_SAPID` will use `sapid` as the key attribute name and `sapcustomer` as the business object name. One column is defined for each key attribute. For static relationships, the column is named `DATA`

The following table shows the show the views in the Common database for the ID relationships.

Table 47. View column definition

Clarify role view	SAP role view	Siebel role view
`INSTANCEID`	`INSTANCEID`	`INSTANCEID`
`CLARIFYCUSTOMER_CLARIFYID`	`SAPCUSTOMER_SAPID`	`SIEBELCUSTOMER_SIEBELID`
`STATUS`	`STATUS`	`STATUS`
`LOGICAL_STATE`	`LOGICAL_STATE`	`LOGICAL_STATE`
`LOGICAL_STATE_TIMESTAMP`	`LOGICAL_STATE_TIMESTAMP`	`LOGICAL_STATE_TIMESTAMP`
`CREATE_TIMESTAMP`	`CREATE_TIMESTAMP`	`CREATE_TIMESTAMP`
`UPDATE_TIMESTAMP`	`UPDATE_TIMESTAMP`	`UPDATE_TIMESTAMP`
`ROLEID`	`ROLEID`	`ROLEID`

Note: All of the column names in the views match, except the key attribute column names.

You must first know the name of the role runtime table view before you can run SQL against the view to manipulate role instance data. The following SQL script

shows an example using DB2 Universal Database™. The example assumes that all the data from each database has been copied to the relationship database. You can copy the data using the SELECT INTO SQL statement:

```
//Create a table to store ID values from all three applications for each customer,
//and associate a unique instance ID with each customer. Use this table as a base
//source table to populate relationship tables.
CREATE TABLE joint_t (instanceid INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,
clarify_id VARCHAR(10) NOT NULL,
sap_id VARCHAR(10) NOT NULL,
siebel_id VARCHAR(10) NOT NULL)

//Compare the name and home phone number across the three application tables.
//If a match is found, insert that person's ID value from each application table
//into the joint_t table. Associate the three ID values to a unique ID; this
//ID will be used later as the relationship instance ID.
INSERT INTO joint_t (clarify_id,sap_id,siebel_id)
SELECT A.ID, B.ID, C.ID
FROM clarifycustomer A,sapcustomer B, siebelcustomer C
WHERE A.homephone=B.homephone AND
B.homephone=C.homephone, AND
B.givenname=C.firstname AND
B.lastname=C.lastname AND
A.fullname=C.firstname CONCAT ' ' CONCAT C.lastname

//Create a sequence for each application; this sequence will be
//used later as a role ID in each role table.
CREATE SEQUENCE clarify_roleid MINVALUE 1 ORDER CACHE 100
CREATE SEQUENCE sap_roleid MINVALUE 1 ORDER CACHE 100
CREATE SEQUENCE siebel_roleid MINVALUE 1 ORDER CACHE 100

//Populate the role instance table for the CLARIFY role.
INSERT INTO V_ID_CLARIFYCUSTOMER_098 (instanceid, roleid,
  clarifycustomer_clarifyid, status, logical_state, logical_state_timestamp,
  create_timestamp, update_timestamp)
FROM joint_t

//Populate the role instance table for the SAP role.
INSERT INTO V_ID_SAPCUSTOMER_515 (instanceid, roleid, sapcustomer_sapid,
  status, logical_state, logical_state_timestamp, create_timestamp,
  update_timestamp)
SELECT instanceid NEXTVAL FOR sap_roleid, sap_id, 0, 0, current
  timestamp, current timestamp, current timestamp
FROM joint_t

//Populate the role instance table for the SIEBEL role.
INSERT INTO V_ID_SIEBELCUSTOMER_AFC (instanceid, roleid, siebelcustomer_siebelid,
  status, logical_state, logical_state_timestamp, create_timestamp, update_timestamp)
SELECT instanceid, NEXTVAL FOR siebel_roleid, sap_id, 0, 0, current timestamp,
  current timestamp, current timestamp
FROM joint_t
```

The joint_t table is created to temporarily store key values. You can delete the table when you are finished to save resources, if necessary. Alternatively, you can create a view table or a temporary table.

Related concepts:

Relationships
Relationships are services used to model and maintain associations between business objects and other data.

"Administering relationships" on page 497
The relationship manager is a tool for manually controlling and manipulating relationship data to correct errors found in automated relationship management or provide more complete relationship information. In particular, it provides a facility for retrieving as well as modifying relationship instance data.

Viewing relationship instances

You can view a list of relationship instances that match the relationship query. The results display in table view and include the relationship instance ID and the property values associated with the instance.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as a monitor, an operator, a configurator, or an administrator.

About this task

To view a list of relationship instances that match the relationship query, complete the steps in the Procedure section:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs (**All**, **By ID**, **By property**, or **By role**) and specify the search criteria. For descriptions of the query options, see "Querying relationships" on page 499.
6. Click **OK** to open the Relationship Instances page.

Results

The list of relationship instances that match your query appears in table view, with each relationship instance shown in its own row. The total page and returned instance counts are displayed at the bottom of the page.

Tip: You can customize the number of rows to display at one time. Click **Preferences**, modify the **row** field value, and click **Apply**. The default is 25, with 1 being the minimum number of records to display at one time and all records being the maximum.

You can navigate through the pages, as follows:

- To view the next set of instances, click the forward arrow.
- To view the previous page of instances, click the back arrow.

Restriction: Filtering or sorting on a large relationship instance count might result in performance problems as it requires getting the full query result set from the server in order to do the sorting. For example, sorting the relationship instance data on a query that would return 20,000 relationship instances needs to sort on those 20,000 instances. The total count (bottom of page) gives an estimate of how many relationship instances you can expect and whether sorting or filtering on a large set of data might lead to long wait times.

For information on setting the query block size parameter to allow for customization of how many instances are read from the server at one time, see the help topic on configuring the relationship service.

Viewing relationship instance details

You can view detailed information for the selected relationship instance, including the relationship name, relationship instance ID, property values, participating roles, and role instance values (role instance ID, logical state, key attributes, and property values). You can view multiple roles concurrently.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as a monitor, an operator, a configurator, or an administrator.

About this task

To view detailed information for the selected relationship instance, complete the steps in the Procedure section:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs (**All**, **By ID**, **By property**, or **By role**); specify the search criteria; and click **OK** to open the Relationship Instances page.
6. You can view the relationship instance details in two ways:
   - Click the relationship instance ID.
   - Select the radio button next to the relationship instance ID and click **Details**.

   To return to the list of relationship instances, click **Relationships Instances** from the path at the top of the page.

Editing relationship instance details

Perform this task to edit the property values for the selected relationship instance.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator to perform this task.

About this task

To edit the property values for the selected relationship instance, complete the steps in the Procedure section.

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs (**All**, **By ID**, **By property**, or **By role**); specify the search criteria; and click **OK** to open the Relationship Instances page.
6. Display the relationship instance details in one of two ways:
   - Click the relationship instance ID.
   - Select the radio button next to the relationship instance ID and click **Details**.
7. Modify the relationship instance property values, as necessary.

   Restriction: You can only edit the property values if they have been previously defined for the relationship instance.
   To delete the relationship instance, click **Delete** at the bottom of the page.

   From this page, you can also create new role instances or delete existing role instances by selecting them and clicking **Create** or **Delete**, respectively, below the role table. Clicking **Create** will open the New Role Instance page for entering key attribute values and property values for the new role instance. You can edit the property values of the role instance by clicking the selected role instance ID.
8. When you are finished making changes in the instance and within the roles of the instance, you have the following options:
   - Click **OK** to save the changes to the system immediately.
   - Click **Cancel** to discard any changes and return to the Relationship Instances page.

Creating new relationship instances

You can create a new relationship instance.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator.

About this task

To create a new relationship instance, do the following:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.

3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Create** to open the New Relationship Instance page.
5. Add the property value information in the **Value** field if you want values other than the default values, and click **OK** to save the new relationship instance locally.

   Note: You must also create a role instance for the relationship instance, as you cannot have a relationship instance without a role instance.

Deleting relationship instances

You can delete a selected relationship instance.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator.

About this task

To delete a selected relationship instance, do the following:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs (**All**, **By ID**, **By property**, or **By role**); specify the search criteria; and click **OK** to open the Relationship Instances page.
6. Select the radio button next to the ID of the relationship instance you want to delete.
7. Click **Delete**.

   The relationship instance is deleted immediately from the system.

Rolling back relationship instance data

You can roll back the relationship instance data to a specified date and time.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator.

About this task

The following actions take place during the rollback:

- Relationship instances which are created during the given period get deleted (hard delete) from the database.
- Relationship instances which are activated get deleted (hard delete) from the database.

- Relationship instances which are deactivated in the given time period get activated.

To roll back the relationship instance data, do the following:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to the relationship services MBean.
4. Select the radio button next to the relationship name and click **Rollback**.
5. Enter the time period for the rollback in the **From date** and **To date** fields.

   Important: Make sure the WebSphere ESB server and the database server are set to the same time zone or the rollback will fail.
6. Click **OK**.

   All instance data in the relationship created later than the specified date and time will be marked as deactivated.

Importing relationships

Perform this task to import data from an existing relationship into your system. Importing an existing relationship is useful in situations when you want to incorporate a relationship from another platform into your solution, but do not want to write code or use relationship manager to add instance details one by one.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator to perform this task.

The relationship manager can import relationships that have been exported in RI or CSV file format. For more information on exporting a relationship instance so it can be used on another platform, see “Exporting relationships” on page 511.

About this task

If there are existing relationship instances in the database, existing relationship instances and newly imported relationship instances will be merged. If the relationship definition that you are importing does not exist, a RelationshipUserException will be thrown.

To import an existing relationship instance, perform the following steps:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications** > **Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Click **Import**.
5. **Browse** for the import file name.

6. Select the radio button next to the correct import file format - either **RI** or **CSV**.
7. Click **OK**.

Results

The relationship instance is imported into the system.

Exporting relationships

Perform this task to export data from an existing relationship to an RI or CSV file. Exporting a relationship is useful in situations when you want to incorporate an existing relationship from one platform into a system running on another platform, but do not want to write code or use relationship manager to add instance details one by one.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator to perform this task.

About this task

To export an existing relationship instance, perform the following steps:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications** > **Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Export**.
5. Optional: Type an export file name. By default, the export file name consists of the relationship name and the extension name (based on your format selection).
6. Select the radio button next to the correct import file format - either **RI** or **CSV** and click **OK**.
7. Select **Save file** and click **OK**.

Results

The relationship instance is exported into an RI or CSV file.

What to do next

Import the RI or CSV file into your solution. For more information on importing a relationship instance so it can be used on another platform, see “Importing relationships” on page 510.

Viewing role instance details

You can view detailed information for the selected role instance, including the role name, role element, key attributes and property values, status, and logical state.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as a monitor, an operator, a configurator, or an administrator.

About this task

To view detailed information for the selected role instance, do the following:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs (**All**, **By ID**, **By property**, or **By role**); specify the search criteria; and click **OK** to open the Relationship Instances page.
6. Display the relationship instance details in one of two ways:
   - Click the relationship instance ID.
   - Select the radio button next to the relationship instance ID and click **Details**.
7. To view the details for the role instance, click its associated ID in the role instance table.

Editing role instance properties

You can edit the property values for the selected role instance.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator.

About this task

To edit the property values for the selected role instance, do the following:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs (**All**, **By ID**, **By property**, or **By role**); specify the search criteria; and click **OK** to open the Relationship Instances page.
6. Display the relationship instance details in one of two ways:
   - Click the relationship instance ID.
   - Select the radio button next to the relationship instance ID and click **Details**.

7. In the role instance table, click the role instance ID to display the role instance details.
8. Edit the role instance property information, as necessary, and click **OK** to save these changes locally.

   Restriction: You can only edit the property values if they have been previously defined for the relationship instance.

Creating new role instances

You can create a new role instance for a relationship.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator.

About this task

To create a new role instance for a relationship, complete the steps in the Procedure section:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs (**All**, **By ID**, **By property**, or **By role**); specify the search criteria; and click **OK** to open the Relationship Instances page.
6. Display the relationship instance details in one of two ways:
   - Click the relationship instance ID.
   - Select the radio button next to the relationship instance ID and click **Details**.
7. Locate the role for which you want to create a new instance and click **Create** below the role table to open the New Role Instance page.
8. Enter the key attribute and role property values in their respective **Value** fields and click **OK** to save the new role instance locally.

   Restriction: You can only set the key attribute value when creating the role instance. You cannot change this information after you have applied the changes back to the database. However, you can edit the property values later.

Deleting role instances

You can delete a selected role instance of a relationship.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as an operator or an administrator.

About this task

To delete a selected role instance of a relationship, complete the steps in the Procedure section:

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Open the relationships page for the server you want to manage by clicking **Relationships** next to that relationship services MBean.
4. Select the radio button next to the relationship name and click **Query**.
5. Click one of the query option tabs (**All**, **By ID**, **By property**, or **By role**); specify the search criteria; and click **OK** to open the Relationship Instances page.
6. Display the relationship instance details in one of two ways:
   - Click the relationship instance ID.
   - Select the radio button next to the relationship instance ID and click **Details**.
7. Locate the role from which you want to delete the role instance.
8. Click the radio button next to the role instance you want to delete and click **Delete** below the role table.

   The role instance is deleted locally.

Removing relationship instance data from the repository

An application that uses relationships has associated relationship schema and data in a repository. The repository is the database configured to hold the relationship instance data. When you uninstall such an application from a production server, the server does not remove the relationship schema and data from the repository. To do so, you need to remove the existing relationship schema manually.

Before you begin

Make sure that you uninstall the application that uses the relationship schema from all servers that access that schema.

About this task

When you install an application containing relationships, the server creates the corresponding database schema objects including tables, indexes, sequences, and stored procedures. At run time, the tables are populated with the relationship instance data. If you uninstall the application that contains relationships, the tables and instance data are not removed from the database. This design can present a problem if you attempt to reinstall the application after modifying the relationship or role definitions.

Tip: If you use the Unit Test Environment (UTE) test server in IBM Integration Designer, the relationship schema and data are removed from the repository when an application project is removed.

If you reinstall the application with the same relationship, the old schema is reused. However, if the relationship or role definition is modified in such a way that makes it incompatible with the existing schema, the relationship service

throws an exception and terminates the installation of the relationship. The logs show the following exception and message:

```
RelationshipServiceException("table <tablename> already exists, but the
table schema is different from current role definition")
```

The solution for this problem is to remove the existing relationship schema artifacts manually, using the tools supplied by the database platform of your repository, and to reinstall the application.

Procedure

To remove the existing relationship schema from the repository, complete the following steps:

1. Locate the database. The database location depends on the database platform.

Option	Description
Database platform	Location
Databases	The location is configured during installation and profile creation of the server. For example, if you configured the server automatically and selected the default database name, the name of the database is WPRCSDB.

2. Delete the database artifacts making up a relationship: Using the tools for your database platform, perform the following steps to delete all database objects for a given relationship.
   a. Before removing any data from the database in the following steps, make a backup of the database.
   b. Find the relationships tables. The following tables are created at the time the relationships are installed.

Table	Format
1 table for relationship properties	`_<relname>_P_uniqueidentifier`
1 table for role properties for each application-specific role	`_<relname>_<rolename>_P_uniqueidentifier`
1 table for each application-specific role (for static relationships 1 table for the generic role is also created)	`_<relname>_<rolename>_RT_uniqueidentifier`

   Restriction: Only the first four characters of the relationship name are used. If the database holds tables for multiple relationships, you should distinguish relationship names within the first 4 characters.

   c. Find the stored procedures. Stored procedure objects have the following format:

      `_<relname>_RS_uniqueidentifier` or
      `_<relname>_<rolename>_RS_uniqueidentifier`

   d. Find the sequences. Sequence objects have the following format:

      `_<relname>_S_uniqueidentifier`

   e. Using the tools for your database platform, delete the following:
      1) tables
      2) stored procedures

3) sequences

Results

The relationship instance data is removed from the database repository.

What to do next

Now you can reinstall the application.

Tutorial: Relationship manager administration

The relationship manager can be used to add, modify, and remove instances of relationships, which correlate identifiers from different environments for the same item of data. This tutorial demonstrates the basic functions of the relationship manager.

This tutorial demonstrates the basic functions of the WebSphere ESB relationship manager. Relationships are used to correlate identifiers from different environments for the same item of data. For example, in one environment, states are identified by two-letter abbreviations (AZ, TX). In another environment, different abbreviations are used (Ariz., Tex.). A relationship would be created to correlate "AZ" in the first environment to "Ariz" in the second environment.

The sample relationship referenced here correlates customer IDs. Many business applications maintain databases of customers, and most of these applications assign their own ID to each customer. In an enterprise environment, the same customer likely has a different ID in each business application. In this tutorial, a relationship is defined to correlate customer IDs. The relationship name is "SampleCustID". Two roles are defined for this relationship. One role is for the Customer Information System (CIS), and the other role is for the General Ledger (GL) application. This relationship was created by the relationship services sample along with the roles and a small amount of sample data.

The relationship manager is designed to add, modify, and remove role instances of a relationship instance as well as add, modify, and remove relationship instances. IBM Integration Designer should be used to create and deploy new relationship definitions. The definitions are stored as XML files that are deployed as part of a Java EE application to a particular server.

Objectives of this tutorial

After completing this tutorial, you will be able to change the values of relationship instances.

Time required to complete this tutorial

This tutorial requires approximately 10 minutes to complete.

Prerequisites

This tutorial uses a relationship that is created by the relationship services technical sample. Before following the steps of this tutorial, go to the samples gallery and perform the steps described in the relationship services sample to create the required relationship and roles.

Example: Changing the values of a relationship instance:

For a relationship instance, the values of key attributes can be changed on the Relationship Instances page of the administrative console. This example shows the use of that page to change a value for a relationship instance.

About this task

One of your customers has a customer ID of A004 in your CIS application. This same customer has a customer ID of 801 in your GL application. However, due to a data entry error, the relationship instance that correlates the customer IDs of this customer currently has a value of 901 instead of 801 for the GL customer ID. This tutorial takes you through the steps to correct this entry in the relationship.

Procedure

1. Open the administrative console.
2. If security is enabled, log in as a user with administrator privileges.
3. In the navigation pane, click **Integration Applications** > **Relationship Manager**.
4. Open the relationships page for the server you want to manage. Click **Relationships** next to that relationship services MBean.

   A relationship named SampleCustID should be visible.
5. Select the radio button next to SampleCustID, then click **Query**.
6. Locate the relationship instance for the customer
   a. Click the query **By role** tab
   b. In the **Role name** field, select `MyGLCustomer_0` from the drop-down list.
   c. In the **Value** field under **Key attributes**, enter`901`
   d. Click **OK**

   This locates the relationship instance for the requested customer and opens the Relationship Instances page.
7. Click the relationship instance ID.

   This displays the relationship instance data for customer ID 901 in the GL application, including all the associated role instances.
8. In the MyGLCustomer_0 role table, select the role instance ID with the key attribute value 901, then click **Delete** below the role table.

   Note: It should not have any property values associated with it. If any other data appears, you need to look at the role instance and record any data you want to keep.
9. Click **Create** to open the New Role Instance page for creating a new role instance for this relationship instance.
10. Enter `801` in the **Value** field under **Key attributes**, then click **OK**.

    The new role instance is saved, and you should see a new role instance in the table.

Results

You now have the correct customer ID value in the relationship instance for the GL application.

Administering the relationship service

The relationship service maintains relationships and roles in the system. It manages relationship and role definitions and metadata and makes it possible to specify the definition of a relationship and manipulate the instances derived from the definition.

The relationship service makes it possible to capture relationships across different objects. Participants in the relationship are distinguished by the roles they serve. For instance, a Person object “Joe” can have an ownership relationship with a Car object “Subaru with license plate XYZ 123.”In this example, Joe participates in the relationship with the role “owner”while the car participates in the relationship under the role “owned object”.

Relationship and role definitions

Relationships and roles are described in definitions that you design through the graphical interface of the relationship editor tool in IBM Integration Designer. The relationship definition is a template that describes what the relationship should look like, identifying the roles each participant in the relationship can assume. The role definition captures the structure and constraint requirements for the participants. Relationship definitions are stored as XML files that are deployed as part of a Java EE application to a particular server.

For detailed background and task information on creating relationships, identifying relationship types, and using the relationship editor, see the IBM Integration Designer Information Center.

How relationships work

At run time, when maps or other WebSphere ESB components run and need a relationship instance, the instances of the relationships are either created or retrieved, depending on the scenario. The relationship and role instance data can be manipulated through three means:

- WebSphere ESB component Java snippet invocations of the relationship service APIs
- Relationship transformations in the WebSphere ESB business object mapping service
- Using the relationship manager tool

The relationship and role instance data is saved in relationship tables that are stored in the database in the default data source that you specify when you configure the relationship service.

The relationship service runs on each server at the cell level. The Relationship Manager home page About section shows the number of servers in the cell that are running relationship services; the Relationships section shows each server name that is running relationship services. Before working with relationship instances, you need to select the server that has the instances of the relationships and roles you want to manage.

Relationship definitions that are contained in user-defined shared libraries will be created in the server at startup time, even if those shared libraries are not associated with any server or application. You can use the relationship manager to manage these relationships.

Removing relationship database artifacts

When you install an application that uses relationships, the relationship service creates many database artifacts, including a set of database artifacts for each relationship and relationship role definition. In a unit test environment (UTE), the relationship database artifacts are removed when you uninstall the application. In a non-UTE environment, the relationship database artifacts are not removed, but are kept unchanged. The next time the application is installed, any new static relationship data in the application is not populated, and if the relationship definition is changed, the relationship table is not recreated. This can result in errors.

To force the relationship database artifacts to be removed, use the RelationshipDatabaseSchemaDrop script or the dropRelationshipDatabaseSchema API before you uninstall the application.

Note: Neither of these interfaces will detect when a relationship is shared by other applications.

For detailed information on using the relationship manager, see the topics on the relationship manager in the WebSphere ESB Information Center.

The following topics describe the configuration tasks to perform for the relationship services for your WebSphere ESB environment.

Viewing relationships managed by the relationship service

Perform this task to view a list of the existing relationships that this relationship service manages.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, any WebSphere security role can view this configuration.

About this task

To view the relationship list, perform the following steps.

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager.**
3. Click **Relationship Services configuration > Relationships**.

   The Relationship collection page displays. Each row shows the version and data source for the associated relationship.

   Tip: To customize the number of rows that display at one time, click **Preferences**. Modify the **Maximum rows** field value and click **Apply**. The default is 25. The total relationship count managed by this relationship service is displayed at the bottom of the page.

What to do next

To see the configuration properties for a relationship, click the relationship name in the relationship collection table.

Viewing relationship properties

Perform this task to view the configuration properties that the relationship service manages at both the relationship service level–as it applies to the relationship service–and at the individual relationship level–as it applies to individual relationships.

Before you begin

Required security role for this task: When security and role-based authorization are enabled, any WebSphere security role can view this configuration.

About this task

To view the configuration properties, perform the following steps.

Procedure

1. Ensure that the administrative console is running.
2. In the navigation pane, click **Integration Applications > Relationship Manager**.
3. Click **Relationship Services configuration > Relationships**.
4. In the relationship collection table, click the name of the relationship with properties you want to view.

   The configuration tabbed page displays, showing the name, version, and data source currently in use for the relationship (read-only).

   Note: The version is used for migration purposes. If the old relationship data needs to coexist in the new system, then the old infrastructure version is set to the old version. Otherwise, it is set to the current version.
5. To return to the Relationship collection page, click **Back**.

RelationshipDatabaseSchemaDrop script

Use the **RelationshipDatabaseSchemaDrop.py** script to remove all relationship database artifacts that are associated with a relationship, including all database artifacts and instance data that are associated with the roles defined for that relationship.

The **RelationshipDatabaseSchemaDrop.py** script is located in the `WPS_HOME/util/RelService` directory.

Make sure that **wsadmin** is connected to the server (in a stand-alone environment) or to the deployment manager (in a Network Deployment environment).

Note: The **RelationshipDatabaseSchemaDrop.py** script causes all the relationship database artifacts that are associated with the relationship to be forcibly removed, even if other applications are using the relationship. The system does not detect shared relationships.

Note: Do not uninstall the application that contains the relationship whose database artifacts you plan to drop before you run the **RelationshipDatabaseSchemaDrop.py** script. Uninstall the application after you run the script.

Required parameter

relationshipFullName
The full name of the relationship, which has the following format: `relationship target namespace` plus "/" plus `relationship short name`. For example, with relationship target namespace `http://RelationshipSample` and relationship short name `CountryRelationship`, the full name would be `http://RelationshipSample/CountryRelationship`.

Example

This example removes all the relationship database artifacts that are associated with relationship full name `http://RelationshipSample/CountryRelationship`.

```
wsadmin -f ${WPS_HOME}/util/RelService/RelServicRelationshipDatabaseSchemaDrop.py http://RelationshipSample/
CountryRelationship
```

Administering loading of mediation flows

The mediation flow component uses lazy loading of resources. The mediation flows, schemas and other resources required to invoke the component, are loaded when the first message that requires processing arrives at the component.

This has the advantage that resources are only loaded when required, which improves server startup time and memory management. The effect of lazy loading is that the first message that arrives at the mediation flow component can take significantly longer to process than subsequent messages. In some cases processes may timeout before the message is successfully processed as the first invocation is disproportionate to the expected processing time.

A WebSphere environment variable is available for such cases where the duration of the first invocation is problematic. It allows for mediation flows and other resources to be loaded at application startup, so when the first message arrives the flow is ready for processing.

The variable is defined using the Integrated Solutions Console, located under **Environment -> WebSphere Variables**. The variable must be defined as following:

```
com.ibm.websphere.sibx.mediation.preload.<moduleName>.<componentName>
```

Where `<moduleName>` is the name of the deployed module and `<componentName>` is the name of the component within that module. The variable's value must be set to an Integer that specifies the number of mediation flow instances that should be preloaded for the specified component.

Wildcards can be used when specifying the module and component name. For example:

`com.ibm.websphere.sibx.mediation.preload.*.*` Will load all components in all modules.

`com.ibm.websphere.sibx.mediation.preload.MyModule.Component*` Will load all components whose name starts with Component in the module MyModule.

`com.ibm.websphere.sibx.mediation.preload.Bank*.*` Will load all components in modules whose name starts with Bank.

The variables can be defined at the server, node, cell or cluster scope with the most specific scope taking priority. When loading an application, if two matching

variables are found, one at the server scope and one at the cell scope, the value of the variable defined at the server scope will take priority.

Administering Common Event Infrastructure

These topics describe several administrative tasks you can use to control the operation of the Common Event Infrastructure components at run time.

Administering the event service using the administrative console

You can use the Web-based administrative console to administer the event service.

About this task

Important: The CEI database is not configured automatically. See Configuring a Common Event Infrastructure (CEI) database for instructions on manually configuring the database.

Enabling and disabling the event service using the administrative console

You can enable and disable the event service by modifying the properties of the event service in the server administrative console.

About this task

If the event service is enabled, it automatically starts when the server starts.

To enable or disable the event service from the administrative console:

Procedure

1. Select one of the following methods to go to the appropriate administrative console panel:
   - Open the Common Event Infrastructure Server panel of the administrative console:
     - For servers, select **Servers** > **Server Types** > **WebSphere application servers** > *server_name* > **Business Integration** > **Common Event Infrastructure** > **Common Event Infrastructure Destination**.
     - For clusters, select **Servers** > **Clusters** > **WebSphere application server clusters** > *cluster_name* > **Business Integration** > **Common Event Infrastructure** > **Common Event Infrastructure Destination**.
   - Alternatively, you can open the Container Services to perform this task:
     - For servers, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name* > **Container Services** > **Common Event Infrastructure Destination**.
     - For clusters, click **Servers** > **Clusters** > **WebSphere application server clusters** > *cluster_name* > **Cluster members** > *server* > **Container Services** > **Common Event Infrastructure Destination**.
2. Select or clear the **Enable service at server startup** property. If the check box is selected, the Common Event Infrastructure service starts when the server starts.
3. Save the configuration changes.
4. In a network deployment environment only, synchronize the node.
5. Restart the servers or cluster.

Creating an event emitter factory using the administrative console

An emitter factory is used by event sources to create emitters.

About this task

The properties of an emitter factory affect the behavior of any emitter that is created using that emitter factory. You can use the default emitter factory or create additional emitter factories for your event sources to use. You might also want to create an additional emitter factory to specify a different transaction mode or event transmission.

To view the event emitter that was created after you configured Common Event Infrastructure (CEI) on the Common Event Infrastructure Destination panel, follow this procedure::

1. Perform one of the following steps:
   - For a single server, select **Servers** > **Server Types** > **WebSphere application servers** > *server_name*.
   - For a cluster, select **Servers** > **Clusters** > **WebSphere application servers** > *cluster_name*.
2. On the **Configuration** tab, select **Business Integration > Common Event Infrastructure > Common Event Infrastructure Destination**.
3. Choose an existing event emitter factory from the menu of JNDI names, or specify one in the text field.

You must resolve JNDI names to a remote server if the CEI server is not local. See the WebSphere Application Server documentation for more information about naming within an ND environment.

Procedure

To create an emitter factory:

1. In the server administrative console, click **Service integration** > **Common Event Infrastructure** > **Event emitter factories** > **New**.
2. Specify the properties of the new emitter factory. Refer to the online help for the Emitter Factory Settings page for detailed information about these properties.
3. Save the configuration changes.
4. In a network deployment environment only, synchronize the node.
5. Restart the server.

Results

Event sources can now use the configured emitter factory to create emitters.

Creating an event group using the administrative console

An event group defines a logical collection of events based on the content of their property data. An event group can be used when querying events from the event service, and it can optionally be associated with a JMS destination for asynchronous event distribution.

About this task

To create an event group:

Procedure

1. Optional: Set up one or more JMS destinations for the event group. An event group can be associated with one JMS topic, and one or more JMS queues. Refer to the documentation for your JMS provider for information about how to create JMS destinations and connection factories and bind them into a JNDI namespace.

   Security: If WebSphere security is enabled, the configuration for the JMS destination must specify an authentication alias.
2. Create the new event group. In the server administrative console, click **Service Integration** > **Common Event Infrastructure** > **Event Service** > **Event Services** > *event_service* > **Event Groups** > **New**.
3. Specify the properties of the new event group, including the event selector and optional JMS destinations.
4. Save the configuration changes.
5. In a network deployment environment only, synchronize the node.
6. Restart the server.

Results

Event consumers can now specify the event group when querying events. If event distribution is enabled in the event service settings, events belonging to the event group are also published to any JMS destinations specified in the event group. Event consumers can then receive events asynchronously by subscribing to the appropriate destinations.

Creating an event filter using the administrative console

An event filter defines properties used by the default filter plug-in, which is used by emitters to filter events at the source.

About this task

An event filter can be specified as part of the configuration of an emitter factory. Any emitter created using that emitter factory then uses the specified filter to determine which event should be sent to the event service.

To create an event filter:

Procedure

1. In the server administrative console, click **Service Integration** > **Common Event Infrastructure** > **Event Emitter Factories** > *emitter_factory* > **Event Filters** > **New**.
2. Specify the properties of the new event filter. Refer to the online help for the Event Filter Settings page for detailed information about these properties.
3. Save the configuration changes.
4. In a network deployment environment only, synchronize the node.
5. Restart the server.

Results

Event emitters can now use the configured filter to determine which events to send to the event service.

Administering the event service using scripting

You can use scripting interfaces to administer the event service.

Enabling the event service using scripting

You can enable the event service by running an AdminTask administrative command using the wsadmin tool.

About this task

If the event service is enabled, it automatically starts when the server starts.

To enable the event service using the wsadmin tool:

Procedure

1. Start the wsadmin tool.
2. Use the AdminTask object to run the **enableEventService** administrative command:
   - Using Jacl:

     ```
     # enable event service at server scope
     $AdminTask enableEventService { -nodeName node1 -serverName server1 }

     # enable event service at cluster scope
     $AdminTask enableEventService  { -clusterName cluster1 }
     ```
   - Using Jython:

     ```
     # enable event service at server scope
     AdminTask.enableEventService('[ -nodeName node1 -serverName server1 ]')

     # enable event service at cluster scope
     AdminTask.enableEventService('[ -clusterName cluster1 ]')
     ```

   The parameters of the **enableEventService** command are as follows:

   nodeName
   The name of the node where the event service should be enabled. This is an optional parameter; the default value is the current node. If you specify a node name, you must also specify a server name. Do not specify a node if you are enabling the event service in a cluster.

   serverName
   The name of the server where the event service should be enabled. This parameter is required if you specify a node name. Do not specify a server name if you are enabling the event service in a cluster.

   clusterName
   The name of the cluster where the event service should be enabled. This parameter is required if you are enabling the event service in a cluster.
3. Restart the server.

Related reference:

enableEventService command
Use the **enableEventService** command to enable the event service on your server.

Disabling the event service using scripting

You can disable the event service by running an AdminTask administrative command using the wsadmin tool.

About this task

If the event service is disabled, it does not automatically start when the server starts.

To disable the event service using the wsadmin tool:

Procedure

1. Start the wsadmin tool.
2. Use the AdminTask object to run the **disableEventService** administrative command:
   - Using Jacl:
     ```
     # disable event service at server scope
     $AdminTask disableEventService { -nodeName node1 -serverName server1 }

     # disable event service at cluster scope
     $AdminTask disableEventService  { -clusterName cluster1 }
     ```
   - Using Jython:
     ```
     # disable event service at server scope
     AdminTask.disableEventService('[ -nodeName node1 -serverName server1 ]')

     # disable event service at cluster scope
     AdminTask.disableEventService('[ -clusterName cluster1 ]')
     ```

   The parameters of the **disableEventService** command are as follows:

   nodeName
   The name of the node where the event service should be disabled. This is an optional parameter; the default value is the current node. If you specify a node name, you must also specify a server name. Do not specify a node if you are disabling the event service in a cluster.

   serverName
   The name of the server where the event service should be disabled. This parameter is required if you specify a node name. Do not specify a server name if you are disabling the event service in a cluster.

   clusterName
   The name of the cluster where the event service should be disabled. This parameter is required if you are disabling the event service in a cluster.
3. Restart the server.

Related reference:

disableEventService command
Use the **disableEventService** command to disable the event service on your server.

Creating an emitter factory with scripting

You can create an emitter factory using a Jacl or Jython script.

Before you begin

Before starting this task, the wsadmin tool must be running.

About this task

Perform the following steps to configure a new emitter factory.

Procedure

1. Identify the Common Event Infrastructure provider ID.
   - Using Jacl:

     ```
     set providerid [$AdminConfig getid \
     /Cell:mycell/Node:mynode/Server:myserver/ \
     EventInfrastructureProvider:/]
     ```

   - Using Jython:

```
providerid =
AdminConfig.getid('/Cell:mycell/Node:mynode/Server:myserver/EventInfrastructureProvider:/')
print providerid
```

     Example output:

```
EventInfrastructureProvider(cells/mycell/nodes/mynode/servers/myserver|resources-cei.xml#
  EventInfrastructureProvider_1)
```

2. Set the required attributes.
   - Using Jacl:

     ```
     set Name [list name "EmitterName"]
     set JndiName [list jndiName "Put JNDI name for new emitter factory here"]
     set Description [list description "Put description here"]
     set Category [list category "Put category here"]

     # set TransactionMode to true to send each event in a new transaction
     set TransactionMode [list preferredTransactionMode "false"]

     # set SynchronizationMode to true to use synchronous event transmission
     # as the preferred synchronization mode
     set SynchronizationMode [list preferredSynchronizationMode "true"]

     # leave blank if synchronous transmission is not supported
     set SyncJNDIName [list synchronousTransmissionProfileJNDIName \
       "Put JNDI name of synchronous transmission profile here "]

     # leave blank if asynchronous transmission is not supported
     set AsyncJNDIName [list asynchronousTransmissionProfileJNDIName \
       "Put JNDI name of asynchronous transmission profile here "]

     set FilteringEnabled [list filteringEnabled "false"]

     # leave blank if filtering is not enabled
     set FilterJNDIName [list filterFactoryJNDIName \
       "Put JNDI name of event filter here"]

     # custom properties include compatibility mode
     set CompatibilityMode [list [list name compatibilityMode] \
       [list description ""] \
       [list required false] \
       [list type java.lang.Boolean] \
       [list value "false"] ] \
     set resProp [list [list resourceProperties [list \
       $CompatibilityMode ]]]
     ```

- Using Jython:

```
Name = ['name', 'EmitterName']
JndiName = ['jndiName', 'Put JNDI name for new emitter factory here']
Description = ['description', 'Put description here']
Category = ['category', 'Put category here']

# set TransactionMode to true to send each event in a new transaction
TransactionMode = ['preferredTransactionMode', 'false']

# set SynchronizationMode to true to use synchronous event transmission
# as the preferred synchronization mode
SynchronizationMode = ['preferredSynchronizationMode', 'true']

# leave blank if synchronous transmission is not supported
SyncJNDIName = ['synchronousTransmissionProfileJNDIName', \
                'Put JNDI name of synchronous transmission profile here']

# leave blank if asynchronous transmission is not supported
AsyncJNDIName = ['asynchronousTransmissionProfileJNDIName', \
                 'Put JNDI name of asynchronous transmission profile here']

FilteringEnabled = ['filteringEnabled', 'false']

# leave blank if filtering is not enabled
FilterJNDIName = ['filterFactoryJNDIName', 'Put JNDI name of event filter here']

# custom properties include compatibility mode
compatibilityName = ['name','compatibilityMode']
compatibilityDescription = ['description','']
compatibilityRequired = ['required', 'false']
compatibilityType = ['type', 'java.lang.Boolean']
compatibilityValue = ['value', 'false']
CompatibilityMode = [compatibilityName, compatibilityDescription, compatibilityRequired, \
                     compatibilityType, compatibilityValue]
customProperties = ['propertySet', [['resourceProperties', [CompatibilityMode]]]]
```

3. Set the properties for the new emitter factory.
   - Using Jacl:

```
set properties "[list $Name $JndiName $Description $Category \
  $TransactionMode $SynchronizationMode $AsyncJNDIName \
  $SyncJNDIName $FilteringEnabled $FilterJNDIName \
              [list propertySet $resProp]]"
```

   - Using Jython:

```
properties = [Name,JndiName,Description,Category,TransactionMode,
SynchronizationMode,AsyncJNDIName,SyncJNDIName,FilteringEnabled,
FilterJNDIName, customProperties]
print properties
```

Example output:

```
[['name', 'EmitterName'], ['jndiName', 'Put JNDI name for new
emitter factory here'], ['description', 'Put description here'],
['category', 'Put category here'], ['preferredTransactionMode',
'false'], ['preferredSynchronizationMode', 'true'],
['asynchronousTransmissionProfileJNDIName', 'Put JNDI name of
asynchronous transmission profile here '],
['synchronousTransmissionProfileJNDIName', 'Put JNDI name of
synchronous transmission profile here '], ['filteringEnabled',
'false'], ['filterFactoryJNDIName', 'Put JNDI name of event
```

```
filter here'], ['propertySet', [['resourceProperties', [[['name',
'compatibilityMode'], ['description', ''], ['required', 'false'],
['type', 'java.lang.Boolean'], ['value', 'false']]]]]]]
```

4. Create the emitter factory.
   - Using Jacl:

     ```
     set emitterProf [$AdminConfig create EmitterFactoryProfile \
     $providerid $properties]
     ```

   - Using Jython:

     ```
     print AdminConfig.create('EmitterFactoryProfile', providerid, properties)
     ```

   Example output:

```
EmitterName(cells/mycell/nodes/mynode/servers/myserver|resources-cei.xml#EmitterFactoryProfile_1)
```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.
7. Restart the server.

Creating an event group with scripting

You can create an event group using a Jacl or Jython script.

Before you begin

Before starting this task, the wsadmin tool must be running. See the WebSphere Application Server documentation for more information.

About this task

Perform the following steps to configure a new event group.

Procedure

1. Identify the Common Event Infrastructure provider ID.
   - Using Jacl:

     ```
     set providerid [$AdminConfig getid \
     /Cell:mycell/Node:mynode/Server:myserver/ \
     EventInfrastructureProvider:/]
     ```

   - Using Jython:

     ```
     providerid =
     AdminConfig.getid
      ('/Cell:mycell/Node:mynode/Server:myserver/EventInfrastructureProvider:/')
      print providerid
     ```

   Example output:

```
EventInfrastructureProvider(cells/mycell/nodes/mynode/servers/myserver|
 resources-cei.xml#EventInfrastructureProvider_1)
```

2. Obtain the event group list.
   - Using Jacl:

     ```
     set eventGroupProfileId [lindex [$AdminConfig list
     EventGroupProfileList $providerid] 0]
     ```

   - Using Jython:

     ```
     eventGroupProfileId = AdminConfig.list('EventGroupProfileList',providerid)
     ```

3. Set the required attributes.
   - Using Jacl:

```
set name [ list eventGroupName "EventGroupName" ]

# escape all '[' characters in the event selector string.
set selectorString [ list eventSelectorString "Set event selector here"]

# leave blank if events should not be published using JMS
set JNDIName [ list topicJNDIName "Set topic JNDI name here"]

# leave blank if events should not be published using JMS
set connectionFactoryJNDIName [ list topicConnectionFactoryJNDIName \
  "Set topic connection factory JNDI name here" ]

set persistEventsFlag [ list persistEvents "true" ]

# custom properties include compatibility mode
set CompatibilityMode [list [list name compatibilityMode] \
  [list description ""] \
  [list required false] \
  [list type java.lang.Boolean] \
  [list value "false"] ] \
set resProp [list [list resourceProperties [list \
  $CompatibilityMode ]]]
```

- Using Jython:

```
Name = ['eventGroupName', 'EventGroupName' ]
SelectorString = ['eventSelectorString', 'Set event selector here']

# leave blank if events should not be published using JMS
JNDIName = [ 'topicJNDIName', 'Set topic JNDI name here']

# leave blank if events should not be published using JMS
ConnectionFactoryJNDIName = ['topicConnectionFactoryJNDIName', \
  'Set topic connection factory JNDI name here']

PersistEventsFlag = ['persistEvents', 'true']

# custom properties include compatibility mode
compatibilityName = ['name','compatibilityMode']
compatibilityDescription = ['description','']
compatibilityRequired = ['required', 'false']
compatibilityType = ['type', 'java.lang.Boolean']
compatibilityValue = ['value', 'false']
CompatibilityMode = [compatibilityName,
                     compatibilityDescription,
                     compatibilityRequired, \
                     compatibilityType,
                     compatibilityValue]
customProperties = ['propertySet', [['resourceProperties',
                    [CompatibilityMode]]]]
```

4. Set the properties for the new event group.
   - Using Jacl:

```
set properties [ list $name $selectorString $JNDIName \
$connectionFactoryJNDIName $persistEventsFlag [list propertySet \
$resProp]]
```

   - Using Jython:

```
properties = [Name,SelectorString,JNDIName,ConnectionFactoryJNDIName, \
              PersistEventsFlag,customProperties]
```

   Example output:

```
[['eventGroupName', 'EventGroupName'], ['eventSelectorString',
'Set event selector here'], ['topicJNDIName', 'Set topic JNDI
name here'], ['topicConnectionFactoryJNDIName', 'Set topic
connection factory JNDI name here'], ['persistEvents','true'],
['propertySet', [['resourceProperties', [[['name',
'compatibilityMode'], ['description', ''], ['required', 'false'],
['type', 'java.lang.Boolean'], ['value', 'false']]]]]]]
```

5. Create the event group.
   - Using Jacl:

```
set result [ $AdminConfig create EventGroupProfile
 $eventGroupProfileId $properties ]
```

   - Using Jython:

```
print AdminConfig.create('EventGroupProfile',
 eventGroupProfileId, properties)
```

   Example output:

```
(cells/mycell/nodes/mynode/servers/myserver|resources-cei.xml#EventGroupProfile_1)
```

6. Save the configuration changes.
7. In a network deployment environment only, synchronize the node.
8. Restart the server.

Creating an event filter with scripting

You can create an event filter using a Jacl or Jython script.

Before you begin

Before starting this task, the wsadmin tool must be running. See the WebSphere Application Server documentation for more information.

About this task

Perform the following steps to configure a new event filter.

Procedure

1. Identify the Common Event Infrastructure provider ID.
   - Using Jacl:

```
set providerid [$AdminConfig getid \
/Cell:mycell/Node:mynode/Server:myserver/ \
EventInfrastructureProvider:/]
```

   - Using Jython:

```
providerid =
AdminConfig.getid('/Cell:mycell/Node:mynode/Server:myserver/EventInfrastructureProvider:/')
print providerid
```

   Example output:

```
EventInfrastructureProvider(cells/mycell/nodes/mynode/servers/myserver|resources-cei.xml#
EventInfrastructureProvider_1)
```

2. Set the required attributes.
   - Using Jacl:

```
set Name [list name "EventFilterName"]
set JndiName [list jndiName "Put JNDI name for new event filter here"]
set Description [list description "Set description of event filter here"]
set Category [list category "Set category for event filter here"]

# escape all '[' characters in the configuration string
```

```
# (for example, "CommonBaseEvent\[@severity=50\]"
set filterConfigurationString [list filterConfigurationString \
  "Set filter configuration string here"]
```

- Using Jython:

```
Name = ['name', 'EventFilterName']
JndiName = ['jndiName',  'Put JNDI name for new event filter here']
Description = ['description', 'Set description of event filter here']
Category = ['category', 'Set category for event filter here']
FilterConfigurationString = ['filterConfigurationString', \
  'Set filter configuration string here']
```

3. Set the properties for the new event filter.
   - Using Jacl:

```
set properties [list $name $jndiName $description $category \
$filterConfigurationString]
```

   - Using Jython:

```
properties = [Name,JndiName,Description,Category,FilterConfigurationString]
print properties
```

   Example output:

```
[['name', 'EventFilterName'], ['jndiName', 'Put JNDI name for new
event filter here'], ['description', 'Set description of event
filter here'], ['category', 'Set category for event filter
here'], ['filterConfigurationString', 'Set filter configuration
string here']]
```

4. Create the event filter.
   - Using Jacl:

```
set filterProf [$AdminConfig create FilterFactoryProfile \
$providerid $properties]
```

   - Using Jython:

```
print AdminConfig.create('FilterFactoryProfile', providerid, properties)
```

   Example output:

```
EventFilterName(cells/mycell/nodes/mynode/servers/myserver|resources-cei.xml#FilterFactoryProfile_1)
```

5. Save the configuration changes.
6. In a network deployment environment only, synchronize the node.
7. Restart the server.

Logging and tracing Common Event Infrastructure components

You can enable logging and tracing in order to debug problems with an application using Common Event Infrastructure.

About this task

Common Event Infrastructure components use the JSR47 Java logging framework, which is available in Common Event Infrastructure server and client environments. For more information about using the logging framework, refer to the WebSphere Application Server Network Deployment, version 6.1 troubleshooting documentation.

The following table indicates the logger names used by the Common Event Infrastructure components.

Table 48. Logger names

Component	Logger name
Root logger name	com.ibm.events
Event catalog	com.ibm.events.catalog
Event service subcomponents	com.ibm.events.access com.ibm.events.bus com.ibm.events.distribution com.ibm.events.server
Default data store plug-in	com.ibm.events.datastore
Event emitter	com.ibm.events.emitter
Notification helper	com.ibm.events.notification
Configuration	com.ibm.events.configuration com.ibm.events.admintask
Database configuration	com.ibm.events.install.db
Migration	com.ibm.events.migration
Miscellaneous utilities	com.ibm.events.util

Maintaining the event database

If you are using an external event database, you should periodically run the provided scripts for database maintenance.

Updating DB2 event database statistics

To enable a DB2 database on Linux, UNIX, or Windows to optimize queries and find free space, you can update the database statistics by running the **runstats** script.

About this task

Updating DB2 database statistics is recommended on a regular basis, and especially under any of the following circumstances:

- Events have been deleted from the database, using either the event deletion interfaces of the event service or the rapid event purge utility of the default data store plug-in
- A large number of events have been inserted into the database
- Tables have been reorganized using the **reorg** script
- Indexes have been added or removed from a table

The **runstats** script is located in the *profile_root*/event/*node_name*/*server_name* or *cluster_name*/dbscripts/db2 directory.

Procedure

To update the database statistics, run one of the following commands:

- Windows On Windows systems:
  `runstats.bat db_user [password=db_password]`
- Linux UNIX On Linux and UNIX systems:
  `runstats.sh db_user [password=db_password]`

The parameters are as follows:

db_user
The database user ID to use. This parameter is required.

db_password
The database password. This parameter is optional; if you do not specify the password on the command line, the DB2 database prompts you for it.

Example

For example, running the following command updates the DB2 database statistics on a Windows system, using the database user ID `dbadmin` and the password `mypassword`:

```
runstats.bat dbadmin mypassword
```

Reorganizing DB2 event database tables

After events are purged or deleted from a DB2 event database on Linux, UNIX, or Windows, you can reorganize the database tables using the **reorg** script.

About this task

The **reorg** script is located in the *profile_root*/event/*node_name*/*server_name* or *cluster_name*/dbscripts/db2 directory.

Procedure

To reorganize the event database tables, run one of the following commands:

- Windows On Windows systems:
  ```
  reorg.bat db_alias db_user [password=db_password]
  ```
- Linux UNIX On Linux and UNIX systems:
  ```
  reorg.sh db_alias db_user [password=db_password]
  ```

The parameters are as follows:

db_alias
The database alias. The event database must be catalogued on the DB2 client; if you are running the script on the DB2 server, the database is already catalogued.

db_user
The database user ID to use. This parameter is required.

db_password
The database password. This parameter is optional; if you do not specify the password on the command line, the DB2 database prompts you for it.

Example

For example, the following command reorganizes the event database tables on a Windows system, using the database alias `eventdb`, user ID `dbadmin` , and the password `mypassword`:

```
reorg.bat eventdb dbadmin mypassword
```

What to do next

After running the **reorg** script, you should update the database statistics using the **runstats** script. For more information, see "Updating DB2 event database statistics" on page 533.

Purging events from the event database

You can run the provided scripts to purge large numbers of events from the event database.

About this task

The default data store plugin provides a set of utilities you can run periodically to purge large numbers of old events from the event database. These utilities are different from the **eventpurge** event service command, which deletes events matching specified criteria.

The database purge capability uses the concept of *buckets*. A bucket is a set of tables used to store events in the event database. The default data store plugin uses two buckets:

- The active bucket is the bucket containing the most recent events; new events are stored in the active bucket. The active bucket cannot be purged using the database purge utility.
- The inactive bucket contains older events. Events stored in the inactive bucket can be queried, deleted, or modified, but typically no new events are stored in the inactive bucket. The inactive bucket can be purged by the database purge utility.

Each event is stored in only one bucket. From the perspective of an event consumer, there is no distinction between the active and inactive buckets; a consumer can query, modify, or delete a specific event without knowing which bucket the event is stored in. The advantage of this approach is that the inactive bucket can be purged using database-specific interfaces without affecting the active bucket; typical event traffic can continue even while the purge operation is taking place.

After the inactive bucket is purged, you can then swap the buckets so that the active bucket becomes inactive and the inactive bucket becomes active. You can swap the buckets only when the inactive bucket is empty.

Although new events are stored only in the active bucket, under some circumstances events might be stored in the inactive bucket immediately after the buckets are swapped. The data store plugin checks periodically to determine which bucket is currently marked as active, but until the next check takes place, some events might continue to be stored in the inactive bucket. Similarly, events sent as part of a batch are all stored in the same bucket, even if that bucket becomes inactive while the batch is still being processed.

If you want to use the fast purge capability, it is your responsibility to determine how frequently to swap buckets or purge the inactive bucket, depending upon event traffic, storage space, archival requirements, or other considerations.

Viewing or changing the event database active bucket status:

The active bucket status indicates which bucket is currently available and which is currently not available.

Procedure

To view or change the active bucket status, use the **eventbucket** command:

```
eventbucket [-status] [-change]
```

This command has the following options:

-status
Use this option to see information about the current bucket configuration, including the active bucket setting and the bucket check interval (the frequency with which the data store plug-in checks to determine which bucket is active).

-change
Use this option to swap the active and inactive buckets. The inactive bucket must be empty before you can use this option.

Purging the event database inactive bucket:

The method used to purge the inactive bucket varies depending on the database software.

Purging the inactive bucket for a DB2 event database (Linux, UNIX, or Windows systems):

On Linux, UNIX, and Windows systems, the database purge utility for a DB2 database is implemented as a shell script or batch file.

Procedure

To purge the inactive bucket, run one of the following commands:

- To purge the inactive bucket, run the following command:
  `fastpurge dbalias dbuser [password=dbpassword] [copydir=copydir]`

The parameters of this command are as follows:

dbalias
The database alias. The event database must be catalogued on the DB2 client; if you are running the script on the DB2 server, the database is already catalogued.

dbuser
The database user ID to use when connecting to the event database.

dbpassword
The password to use for the specified user ID. This parameter is optional; if you do not specify the password, the DB2 database prompts you for it.

copydir
The path to a directory to be used for the files generated by the load utility. This parameter is required only if you have enabled forward recovery for the event database (with the LOGRETAIN or USEREXIT database configuration settings enabled). By default, the event database does not use forward recovery.

Purging the inactive bucket for a DB2 event database:

The database purge utility for a DB2 event database is implemented using the DB2 load utility.

About this task

To purge the inactive bucket:

Procedure

1. Run the **eventbucket** command to identify the inactive bucket (bucket 0 or bucket 1).
2. Upload the appropriate utility control file. These files are generated during database configuration and are located in the /event/dbscripts/db2zos directory. Upload one of the following files:
   - If bucket 0 is inactive, upload the fastpurge00.ctl file.
   - If bucket 1 is inactive, upload the fastpurge01.ctl file.

   Upload format: The control file must be uploaded with a fixed record format and a logical record length of 80.
3. Go to the ISPF DB2I Primary Option Menu and select the **Utilities** option.
4. Specify the following information:

Field	Value
Function	EDITJCL
Utility	LOAD
Statement Data Set	The name of the data set containing the uploaded control file
LISTDEF	NO
Template	NO

5. Press Enter to continue to the next panel.
6. In the recdsn entry field, type the name of the data set containing the uploaded control file.
7. Press Enter. The JCL script to purge the inactive bucket is generated.
8. Press Enter to clear the output messages.
9. Edit the generated JCL script as needed.
10. Submit the JCL script.

Results

The generated JCL script can be reused. After you create the scripts for purging both buckets (0 and 1), you do not need to repeat this entire procedure.

Purging the inactive bucket for an Oracle event database:

The database purge utility for an Oracle event database is implemented as a stored procedure.

Before you begin

Before you can use the database purge utility for an Oracle database, SQL*Plus must be installed on the Oracle client, and the client must be configured to communicate with the Oracle database (the tnsnames.ora file must be configured properly). Refer to the Oracle documentation for more information.

Procedure

To purge the inactive bucket, run the stored procedure using SQL*Plus:

```
sqlplus connect_string @fastpurge.sql
```

The fastpurge.sql file is stored in the same location as the other scripts generated during database configuration (the default location is *profile_root*/databases/event/*node_name*/*server_name* or *cluster_name*/dbscripts/oracle).
The *connect_string* parameter is the Oracle connection string. Use the same database user ID used to create the event database.

Purging the inactive bucket for a SQL Server event database:

The database purge utility for a SQL Server event database is implemented as a stored procedure.

Before you begin

To use the utility for a SQL Server database, you must have the osql utility installed on the SQL Server client. Refer to the SQL Server documentation for more information.

Note: Do not use the isql utility for running the database purge stored procedure.

Procedure

To purge the inactive bucket, run the stored procedure using the osql utility:

```
osql -Sserver_name -Udbuser -Pdbpassword -deventdb -Q"fast_purge"
```

The fast_purge file is stored in the same location as the other scripts generated during database configuration (the default location is *profile_root*/databases/event/*node*/*server*/dbscripts/sqlserver).
The parameters of this command are as follows:

server_name
: The database server name.

dbuser
: The database user ID to use when connecting to the event database.

dbpassword
: The password to use for the specified user ID. This parameter is optional; if you do not specify the password, the osql utility prompts you for it.

eventdb
: The name of the event database (typically `event`).

Changing the event database bucket check interval:

The bucket check interval specifies how frequently the data store plug-in checks to determine which bucket is active. This value is specified as a custom property in the data store settings.

About this task

The default bucket check interval is 5 minutes (300 seconds). A shorter interval reduces the likelihood that events will be stored in the inactive bucket after swapping, but might decrease performance.

To change the bucket check interval:

Procedure

1. In the administrative console, click **Service integration** > **Common Event Infrastructure** > **Event service** > **Event services** > *event_service* > **Event data store**.
2. Modify the value of the BucketCheckInterval property to specify the bucket check interval in seconds.

Results

The change takes effect the next time the server is started.

Changing the configuration topology

You can change the WebSphere ESB configuration topology by moving components from one topology to another topology that better addresses the processing requirements of your business.

Through tuning exercises, an administrator may determine that the current WebSphere ESB configuration topology needs to be changed in order to meet performance requirements.

This may mean moving the components of a single cluster topology to a multi-clustered topology (remote messaging and support for example) to better manage the potential system bottlenecks.

One reason for changing your configuration topology might be to more evenly distribute processing load. For example, if you have installed everything on a single cluster and you now want to have a separate cluster for your support applications to reduce the load on your application cluster, you could move some of the components currently on the application cluster to the support cluster. Another reason for changing the configuration might be that you do not need high availability for the applications on the support cluster.

Remember, a topology represents the configuration of the functional components in your environment.

To move from one topology to another, you must move the individual components that make up the WebSphere ESB configuration from its current topology, to a new topology.

You can change the WebSphere ESB configuration from a command line, using wsadmin.

Before changing a configuration topology, stop processing any events and make sure there are no messages in queues and in the database.

Changing the configuration topology using the command line

You can use the command line (wsadmin) to change your WebSphere ESB configuration.

Using the command line to move the CEI server to a different deployment target

You can move the Common Event Infrastructure (CEI) server from one deployment target to another using the **wsadmin** command.

Before you begin

Understand the business rationale for moving the CEI server.

You must be at the deployment manager from which you will move the CEI server.

Required security role for this task: When security and role-based authorization are enabled, you must use a user ID and password with administrator or operator authority to perform this task.

About this task

This task describes how to use the **wsadmin** command to move the CEI server from one deployment target (either a server or a cluster) to another deployment target.

You might need to move the CEI server if you previously configured a Single Cluster topology in which the CEI server was configured in the application cluster, and because of new processing requirements, you need to move it to the support cluster of a Remote Messaging and Remote Support topology.

In some instances, having the CEI server on the application cluster might adversely affect the processing capabilities of applications deployed on the application cluster. Moving the CEI server off the application cluster and on to a support cluster can reduce the adverse effect that CEI processing has on deployed applications.

Procedure

1. Open a command window.

   The wsadmin command can be found at `<install_root>/bin`. Use the wsadmin `-profileName` parameter to specify the profile to work with.
2. At the command prompt, enter the **wsadmin** command to enter the **wsadmin** environment.
3. Use the **moveCEIServer** command to move the CEI server from the current deployment target (cluster or server) to a new deployment target (cluster or server).

Example

The example moves the CEI server from one cluster to another cluster. The example is shown for both Jacl and Jython.

Note: If you are running the admin client from the deployment manager `bin` folder, you do not need to include the `-host` and `-port` parameters in the command.

Using Jacl

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask moveCEIServer { -fromClusterName Cluster1 -toClusterName Cluster2
> $AdminConfig save
```

Using Jython

```
wsadmin -lang jython -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> AdminTask.moveCEIServer('[-fromClusterName cluster1 -toClusterName cluster2]')
> AdminConfig.save()
```

This example moves the CEI server from one single server configuration to another single server configuration. The example is shown for both Jacl and Jython

Using Jacl

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask moveCEIServer { -fromServerName Server1 -toServerName Server2 -fromNodeName Node1
-toNodeName Node2}
> $AdminConfig save
```

Using Jython

```
wsadmin -lang jython -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> AdminTask.moveCEIServer('[-fromServerName Server1 -toServerName Server2 -fromNodeName Node1
-toNodeName Node2]')
> AdminConfig.save()
```

What to do next

You can use the administrative console to verify that the CEI server has been moved. The application should now be mapped to the new deployment target.

Note: The previous deployment target is not changed other than the fact that the CEI server has been removed.

Using the command line to determine the target significance setting of deployed applications

You can use wsadmin to discover the target significance setting used for all the activation specification and connection factories configured on the given cluster or server.

Before you begin

You must be at the deployment manager from which you will discover target significance.

Required security role for this task: When security and role-based authorization are enabled, you must use a userid and password with administrator or operator authority to perform this task.

.

About this task

This task describes how to use wsadmin to discover the target significance setting used for all the activation specification and connection factories configured on the given cluster or server.

Run this command to determine the current setting for `targetSignificance`. The value for `targetSignificance` is either **Required** or **Preferred** or **unknown**.

A value of **unknown** is returned if some of the resources have a setting of **Required**, while other resources have a value of **Preferred**. Use the **-printDetails** option to list the target significance settings. You can change the target significance settings by running the `setBPMTargetSignificance` command.

Procedure

1. Open a command window.

The wsadmin command can be found at `<install_root>/bin`. Use wsadmin's `-profileName` parameter to specify the profile to work with.

2. At the command prompt, enter the **wsadmin** command to enter the **wsadmin** environment.
3. Use the **getBPMTargetSignificance** command to retrieve the target significance setting used for all the activation specification and connection factories configured on the given cluster or server.

Example

The example discovers the target significance settings for all activation specifications and connection factories on a server. The example is shown for both Jacl and Jython. In the example, the -printDetails property is set to true. If you do not want to print the target significance settings for all the activation specification and connection factories configured on the given cluster or server, set -printDetails to **false**.

Using Jacl

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask getBPMTargetSignificance { -serverName serverName -nodeName nodeName -printDetails true}
> $AdminConfig save
```

Using Jython

```
wsadmin -lang jython -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> AdminTask.getBPMTargetSignificance ('[-serverName serverName -nodeName nodeName -printDetails true ]')
> AdminConfig.save()
```

The example discovers the target significance for all activation specifications and connection factories on a cluster. The example is shown for both Jacl and Jython. In the example, the -printDetails property is set to true. If you do not want to print the target significance settings for all the activation specification and connection factories configured on the given cluster or server, set -printDetails to **false**.

Using Jacl

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask getBPMTargetSignificance { -clusterName clusterName -printDetails true}
> $AdminConfig save
```

Using Jython

```
> AdminTask.getBPMTargetSignificance ('[-clusterName clusterName -printDetails true]')
> AdminConfig.save()
```

What to do next

If **-printDetails=true**, the command prints the current setting for `targetSignificance` used for all the activation specification and connection factories configured on the given cluster or server.

Now that you have discovered the target significance settings for the activation specifications and connection factories, you can use the `setBPMTargetSignificance` command to change the value assigned to target significance if necessary.

For more information on how `targetSignificance` values affect processing, see the WebSphere Application Server Information Center.

Using the command line to change how applications interact with bus members

You can use wsadmin to change the way applications interact with the service integration bus to select a target bus member and messaging engine by changing the target significance value assigned to the application.

Before you begin

Retrieve the current target significance used for all the activation specification and connection factories configured on the given cluster or server by running the command getBPMTargetSignificance.

About this task

This task describes how to use wsadmin to set the target significance setting used for all the outbound connection factories and for the activation specifications (the outbound modules, via the module import) configured on the given cluster or server.

The target significance value determines how the applications interact with bus members defined to a messaging engine (ME).

Remember: The system-generated Java Message Service (JMS) connection factory (named as `<moduleName>/<compName>_CF` is shared for inbound and outbound. As a result, the target significance setting for the JMS connection factory is not changed when you run **setBPMTargetSignificance**. For the JMS, the connection factory used for Failed events (`<moduleName><compName>/_FAILEDEVENT_REPLAY_CF`) is changed when your run **setBPMTargetSignificance**.

An Activation Specification provides the necessary configuration details for a message driven bean (MDB) and includes information on where to connect, and from which destination to consume messages. An activation specification may be created by the developer or administrator, or automatically by the SCA infrastructure as part of the definition of an SCA module. In the case where the messaging engine hosting the destination is not running, the MDB will not be able to consume messages even if it connects to a different messaging engine, as that ME will still need to contact the unavailable ME to retrieve the messages. So for activation specifications we should always set the significance to **Required** for best efficiency.

A connection factory can be used to send or receive messages, and the destination(s) it will use are not known at configuration time. If the administrator knows that a particular connection factory will only be used to consume messages then the same logic applies as above for activation specifications, and a **Required** target should be set for the bus member that localizes the destination. If the connection factory will only be used to send messages, then it is usually desirable to set the significance to **Preferred** in order to have better fault tolerance (the message will be sent if any messaging engine in the bus is available). Two exceptions to this are the following cases, which require a setting of **Required** to be used:

- Manageability and serviceability
- Message ordering, or Event Sequencing. If the **Preferred** setting is used then an application might send messages through different paths if failures occur, which

makes it impossible to guarantee the ordering of the messages as they arrive at the destination. If message ordering is a requirement then **Required** should always be used.

Procedure

1. Open a command window.

   The wsadmin command can be found at `<install_root>/bin`. Use wsadmin's `-profileName` parameter to specify the profile to work with.
2. At the command prompt, enter the **wsadmin** command to enter the **wsadmin** environment.
3. Use the **setBPMTargetSignificance** command to set the target significance value for the activation specification and connection factories configured on the given cluster or server.

Example

The example sets the target significance value of *Required* for all activation specifications and connection factories on a server. The example is shown for both Jacl and Jython. In the example, the -printDetails property is set to true. If you do not want to print the target significance settings for all the activation specification and connection factories configured on the given cluster or server, set -printDetails to **false**.

Using Jacl

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask setBPMTargetSignificance { -serverName serverName -nodeName nodeName
-targetSignificance Required -printDetails true}

$AdminConfig save
```

Using Jython

```
AdminTask.setBPMTargetSignificance ('[-serverName serverName -nodeName nodeName
-targetSignificance Required -printDetails true]')

AdminConfig.save()
```

The example sets the target significance value of *Required* for all activation specifications and connection factories on a cluster. The example is shown for both Jacl and Jython. In the example, the -printDetails property is set to true. If you do not want to print the target significance settings for all the activation specification and connection factories configured on the given cluster or server, set -printDetails to **false**.

Using Jacl

```
wsadmin -connType SOAP -host myDmgr -port 8879 -user dmgradmin -password dmgrpass
> $AdminTask setBPMTargetSignificance { -clusterName clusterName -targetSignificance Required
-printDetails true}

$AdminConfig save
```

Using Jython

```
AdminTask.setBPMTargetSignificance ('[-clusterName clusterName -targetSignificance Required
-printDetails true]')

AdminConfig.save()
```

What to do next

You can run the `getBPMTargetSignificance` command to verify the target significance settings.

Working with adapters

WebSphere ESB supports two types of adapters: WebSphere Adapters and WebSphere Business Integration Adapters. Adapters enable business applications to act as services by connecting them to diverse enterprise information systems (EISs), such as databases, enterprise resource planning systems, file systems, and e-mail systems.

With the help of an adapter, the application and EIS can communicate, sending and retrieving information in a consistent way. To allow your applications to operate as services, the adapter connects them to WebSphere ESB, which powers your Service Oriented Architecture (SOA). With an adapter, you no longer need to provide proprietary connection utilities (or write custom connection utilities) for each EIS or application server.

Differences between WebSphere Adapters and WebSphere Business Integration Adapters

Both WebSphere Adapters and WebSphere Business Integration Adapters mediate communication between components and enterprise information systems. The two types of adapter differ in several respects including: their integration, their JCA-compliance, their data models, and the management of their connectivity.

There are several differences between WebSphere Adapters and WebSphere Business Integration Adapters. These distinctions are most important during development of applications. When deploying applications to a running server, the nature of the adapters used affects some of the steps that need to be followed.

Adapters provide communication mechanisms between enterprise information systems (EISs) and Integration Designer applications. To illustrate the operation of the adapters, Figure 135 on page 546 and Figure 136 on page 547 provide details of the communication between the server and the EIS for the two types of adapters.

Figure 135. Detailed schematic of a WebSphere Adapter

Figure 135 depicts a WebSphere Adapter managing the connectivity between a Java EE component supported by the server and the EIS. The WebSphere Adapter resides inside the server.

Figure 136. Detailed schematic of a WebSphere Business Integration Adapter.

Figure 136 shows a WebSphere Business Integration Adapter mediating communication between the WebSphere Integration Broker and the EIS. The integration broker communicates with the WebSphere Business Integration Adapter through the use of a Java Message Service (JMS) transport layer.

Table 49 shows the differences between the two types of adapter.

Table 49. Differences between WebSphere Adapters and WebSphere Business Integration Adapters

Feature	WebSphere Adapters	WebSphere Business Integration Adapters
JCA Compliance	Fully JCA compliant (version 1.5).	Not JCA-compliant.
Connectivity Manager	Rely on standard JCA contracts to manage life cycle tasks such as starting and stopping.	Rely on WebSphere Adapter Framework to manage connectivity.
Event Notification	Use an EventStore subclass to retrieve events from an EIS.	Manage event notification using a pollFor Events method.
Request Processing	Clients directly invoke one of several interaction contracts to query or modify data in the EIS.	Rely on an integration server and the WebSphere Adapter Framework to initiate and help process requests.
Data Models	Use an Enterprise Metadata Discovery (EMD) utility to parse an EIS and develop Service Data Objects (SDOs) and other useful artifacts. The EMD is part of the WebSphere Adapter implementation.	Use a separate Object Discovery Agent (ODA) to introspect an EIS and generate business object definition schemas.

Table 49. Differences between WebSphere Adapters and WebSphere Business Integration Adapters (continued)

Feature	WebSphere Adapters	WebSphere Business Integration Adapters
Integration	Run on the server.	Reside outside the server. The server or integration broker communicates with the adapter via a Java Message Service (JMS) transport layer.

WebSphere Adapters are the recommended product.

WebSphere adapters

WebSphere adapters, also known as resource adapters, enable managed, bidirectional connectivity between enterprise information systems (EISs) and Java EE components supported by the server.

WebSphere adapters, which are preferred over WebSphere Business Integration adapters, are covered elsewhere in this information library.

Where to find more information

To learn about configuring and using WebSphere adapters, see Configuring and using adapters. In the adapter guide for your adapter, expand the navigation and click **Administering the adapter module**.

For general information about adapters, see Accessing external services with adapters.

For information about EIS bindings, which are used with WebSphere adapters to provide connectivity between SCA components and an EIS, see "EIS bindings" on page 116.

WebSphere Business Integration Adapters

WebSphere Business Integration Adapters consist of a collection of software, Application Programming Interfaces (APIs), and tools to enable applications to exchange business data through an integration broker.

Each business application requires its own application-specific adapter to participate in the business integration process. You can install, configure, and test the adapter using current WebSphere Business Integration Adapter Framework and Development Kit System Manager tools. You can use IBM Integration Designer to import existing business objects and connector configuration files, to generate artifacts, and to assemble the solution for WebSphere ESB.

Operational commands for the WebSphere Business Integration Adapters are part of the administrative console.

Where to find more information

For more information about working with these adapters, see Using WebSphere Business Integration Adapters.

Managing the WebSphere Business Integration Adapter

You can manage a running WebSphere Business Integration Adapter from the administrative console.

Before you begin

The WebSphere Business Integration Adapter must be running in order to be managed.

About this task

Use the following procedures to manage your resources and to perform various administrative actions on them.

Procedure

1. Select the resource or resources to manage. From the top level of the administrative console, follow these steps:
   a. Expand **Servers**.
   b. Expand **Server types**.
   c. Select **WebSphere application servers**.
   d. From the list of servers, select the server where the resources you intend to manage reside.

      Click on the name of the server that hosts the resources of interest.
   e. From the **Business Integration** list on the Configuration tab, select **WebSphere Business Integration Adapter Service**.
   f. Select **Manage the WebSphere Business Integration Adapter resources**.
   g. From the list of resources, select the check boxes associated with the resources you intend to manage.
2. Manage the selected resources.

   Click one of the command buttons to act upon the selected resources.

Command	Description
Deactivate	Changes the status of the selected resources from active to paused or inactive.
Activate	Changes the status of the selected resources from inactive to active.
Suspend	Changes the status of the selected resources from active to paused.
Resume	Changes the status of the selected resources from paused to active.
Shut down	Changes the status of the selected resources from active to unavailable.

Working with events

Events are requests or responses sent from one component to another. You can process events in a specific sequence. When events fail, you can use the failed event manager to view, discard, modify, or resubmit the events.You can also use the Store and forward feature to prevent subsequent failures from occurring when a component calls asynchronously to a service that is unavailable.

Processing events in sequence

Event sequencing enables IBM Process Server and WebSphere Enterprise Service Bus components to process events from asynchronous invocations in the order in which they are delivered. Event order is maintained throughout the entire business integration scenario.

An *event* is a request or a response that is sent from one component to another. The event encapsulates data and invocation metadata (for example, the name of the target component, the operation, and the parameters).

Note: Event sequencing is supported only for requests sent with an asynchronous invocation.

Why use event sequencing?

Some implementations require the target component to process events in the same order in which they were sent by the source application; processing them out of order can cause errors or exceptions. For example, if a source application generates an event to create a business object and then generates an event to update that business object, the create event must be processed first.

In an asynchronous invocation, events are stored in destinations on a service integration bus and can be handled by multiple instances of Message Driven Beans (MDBs). As a result, they may be processed non-sequentially, which can cause failures. To avoid this problem, use event sequencing.

How does event sequencing work?

Enable event sequencing by using the *event sequencing qualifiers* available in Integration Designer. The qualifiers must be set on each method that requires event sequencing; they tell the runtime environment that invocations to these methods need to be sequenced.

Each qualifier has an event sequencing key that determines how events are sequenced. The key's value consists of one or more attributes of the business objects associated with an invocation. All events that share the same key are grouped together and processed in sequence. Events that do not have an event sequencing key continue to be processed as normal, in parallel with the sequenced events.

A sequenced event acquires a lock before being sent to the target component for processing. As soon as the business logic for the event has executed, the lock is released and given to the next event with the same event sequencing key. If the event cannot acquire the necessary lock, the execution of the invocation is suspended until the lock is acquired.

Related concepts:

"Example: Event sequencing"
To understand how event sequencing works, consider a situation in which a source application (Component A) asynchronously invokes a target application (Component B) to create new orders, and then updates those orders with revised data.

"Considerations for implementing event sequencing" on page 553
Use the information in these topics to help you plan for, implement, and troubleshoot event sequencing in your business integration environment.

Example: Event sequencing

To understand how event sequencing works, consider a situation in which a source application (Component A) asynchronously invokes a target application (Component B) to create new orders, and then updates those orders with revised data.

Component A looks up Component B and invokes the create method to create an order, using the Order business object. The Order business object has the following attributes:

Attribute	Type
ID	string
customer	string
productName	string
quantity	integer

Component A then calls the update method to update the data in the newly created order.

In this example, assume there are five separate events that have been sent from Component A to Component B in the order specified below:

- Create1: This invocation calls the create method and passes the Order business object with an ID of 1 and quantity of 10.
- Create2: This invocation calls the create method and passes the Order business object with an ID of 2 and a quantity of 8.
- Update1: This invocation calls the update method and passes the Order business object with an ID of 1 and a quantity of 15.
- Update2: The third invocation calls the update method and passes the Order business object with an ID of 1 and a quantity of 12.
- Update3: This invocation calls the update method and passes the Order business object with an ID of 2 and a quantity of 10.

For each event, a message is put onto a service integration bus destination in the same order as the invocations. A Message Driven Bean (MDB) reads the message and sends it to the target component (in this case, Component B) for processing. Although there is only one MDB per module, there are multiple instances of that MDB and these five messages are processed in parallel. It is possible that the MDB thread that is processing the message for Update2 will complete before the thread that is processing the message for the Create1 event; if this happens, the Update2 event will fail because the order has not yet been created.

To prevent these sorts of errors, this example implements event sequencing. In the sample component definition below, event sequencing qualifiers are specified for

both the create and update methods. Both of these methods use the same event sequencing key (set to the ID attribute of the Order business object) and are placed into the same event sequencing group. The third method, retrieve, is not sequenced.

```
<interfaces>
   <interface xsi:type="wsdl:WSDLPortType" portType="ns1:ProcessOrder">
      <method name="create">
         <scdl:interfaceQualifier xsi:type="es.EventSequencingQualifier">
         <es:eventSequencing sequencingGroup="default">
            <keySpecification>
               <parameter name="Order">
                  <xpath>ID</xpath>
               </parameter>
            </keySpecification>
         </es:eventSequencing>
         </scdl:interfaceQualifier>
      </method>
      <method name="update"/>
         <scdl:interfaceQualifier xsi:type="es:EventSequencingQualifier">
         <es.eventSequencing sequencingGroup="default">
            <keySpecification>
               <parameter name="Order">
                  <xpath>ID</xpath>
               </parameter>
            </keySpecification>
         </es:eventSequencing>
         </scdl:interfaceQualifier>
      <method name="retrieve/">
   </interface>
</interfaces>
```

With event sequencing enabled, the five events in this example are processed as follows:

1. Component A sends the Create1 request. It is placed on the destination and handled by an instance of the MDB.
2. The Create1 event acquires a lock and is sent to Component B for processing.
3. Component A sends the Update1 request. It is placed on the destination and handled by an instance of the MDB.
4. The Update1 event tries to acquire a lock. If the Create1 event (which shares the same event sequencing key value as Update1) still has the lock, processing for this event is suspended until the lock on Create1 is released.
5. Component A sends the Create2 request. It is placed on the destination and handled by an instance of the MDB.
6. The Create2 request (which has a different value for the event sequencing key) acquires a lock and is sent to Component B for processing.
7. Component A sends the Update2 request. It is placed on the destination and handled by an instance of the MDB.
8. The Update2 event tries to acquire a lock. If either the Create1 or Update1 event (which share the same event sequencing key value as Update2) still holds a lock, processing for this event is suspended. It will not be processed until the Update1 event has acquired the lock, been processed, and the lock has been released.
9. Component A sends the Update3 request. If the Create2 event (which shares the same event sequencing key value as Update3) still has the lock, processing for this event is suspended until the lock on Create2 is released.

Related concepts:

"Processing events in sequence" on page 550
Event sequencing enables IBM Process Server and WebSphere Enterprise Service Bus components to process events from asynchronous invocations in the order in which they are delivered. Event order is maintained throughout the entire business integration scenario.

Considerations for implementing event sequencing

Use the information in these topics to help you plan for, implement, and troubleshoot event sequencing in your business integration environment.

Related concepts:

"Processing events in sequence" on page 550
Event sequencing enables IBM Process Server and WebSphere Enterprise Service Bus components to process events from asynchronous invocations in the order in which they are delivered. Event order is maintained throughout the entire business integration scenario.

Supported components and invocations:

Before you implement event sequencing, consider the types of invocations and components you are using and whether they support sequencing.

Event sequencing is supported for all requests from Service Component Architecture (SCA) components that meet the following requirements:

- Components must use Web Services Description Language (WSDL) interfaces.
- Components must use asynchronous invocation.

  Note: The client is responsible for maintaining event order before events are put on SCA destinations. If sequencing is required, the client must do the SCA invocations within a single thread.

It is not supported for responses.

You do not need to use event sequencing for events that are implicitly sequenced during a synchronous invocation to a component with a synchronous implementation. If the client is using a single thread for invocations, the call automatically waits until the target has finished processing the event. No further invocations can be made until the event is returned.

Event sequencing declarations for components:

After you have determined which methods on a component need to use event sequencing, use Integration Designer to update the component definition to include an event sequencing qualifier to the each of those methods.

Important: When declaring event sequencing on a component, ensure that the component is invoked in a managed thread. The managed thread provides the session information required to properly sequence events.

Event sequencing qualifiers extend types defined in the Service Component Definition Language (SCDL), enhancing the quality of service for Service Component Architecture (SCA) components.

The event sequencing qualifier contains a keySpecification element to identify the events to sequence. There must be one keySpecification element for each method

that uses event sequencing. The parameter element is used with each keySpecification; it indicates the business object attribute or attributes that will provide the value for the event sequencing key.

Use the event sequencing qualifier's attributes to further extend sequencing functionality. For example, the sequencingGroup attribute groups methods that need to be sequenced together; all events that are generated by any method in the same group are processed sequentially.

IBM Integration Designer provides a setting to determine the event processing behavior of sequenced events when encountering unexpected runtime failures when authoring the Event Sequencing qualifier for the specific service operation. The configuration setting for the qualifier is controlled by the check box called “Process requests when error encountered”. For further information about the configurable behavior of failed event processing of sequenced events, see the topic devoted to Failed Sequenced Events in this documentation.

When declaring event sequencing on a component, ensure that the component is invoked in a managed thread. The managed thread provides the session information required to properly sequence events.

Event sequencing with export bindings:

Event sequencing is supported with EIS, JMS, WebSphere MQ, and WebSphere MQ JMS export bindings. To ensure that the exports process and deliver messages in the correct sequence, you must configure the export bindings appropriately.

Consider the following requirements when using event sequencing on a target component that handles export bindings:

- An adapter component must use the non-optimized path for a Java Message Service (JMS) export when event sequencing is used on the target component.
- To enable event sequencing for JMS export bindings, you must limit the number of concurrent Message Driven Beans (MDBs) that are processing incoming messages. Do this by setting the maxConcurrency custom property on the ActivationSpec to a value of 1.
- To enable event sequencing for a WebSphere MQ JMS export , you must limit the number of concurrent listener threads that will deliver messages to the Message Driven Bean. Do this by setting the maxSessions property to a value of 1.
- To enable event sequencing for a native MQ export, you must use Integration Designer to set the eventSequencing property.

Related information:

Enabling event sequencing for an EIS Export binding
WebSphere adapters provide a mechanism to allow event sequencing in WebSphere ESB by specifying an activation specification property. Also, the export must process and deliver messages in the same order it received those messages.

Enabling event sequencing for a JMS Export binding
To enable event sequencing for JMS export bindings in WebSphere ESB, you must configure properties of the binding. Also, the export must process and deliver messages in the same order that it receives those messages.

Enabling event sequencing for a WebSphere MQ JMS Export bindings
To enable event sequencing for WebSphere MQ JMS export bindings in WebSphere ESB, you must configure properties of the binding. Also, the export must process and deliver messages in the same order that it receives those messages.

Event sequencing with store-and-forward:

Setting an event sequencing qualifier and a store-and-forward qualifier on the same interface is supported. When enabled simultaneously, event sequencing is stopped when the store is started by a qualifying runtime exception. The sequenced processing of events resumes when the store is stopped and forwarding is started.

When store-and-forward is in store mode, the “Process requests when error encountered” attribute of the event sequencing qualifier is ignored.

Important: For failed events that are sequenced, you must change the store-and-forward state to forward in order to resubmit events. Events cannot be resubmitted when store-and-forward is in store mode.

For network deployment environments where an application cluster has two or more members and both event sequencing and store-and-forward qualifiers are specified, the store-and-forward service is enabled for all members of the cluster. When an event triggers the store, event-sequencing processing stops.

For example, consider the following situation. Event sequencing and store-and-forward qualifiers are declared for failed events being processed by a cluster with two members, called member A and member B. The event-sequencing service processes events on member A of the cluster. When the store is triggered by a qualifying runtime exception, event sequencing processing stops . If member A becomes unavailable, event sequencing is activated on member B. On member B, event sequencing processing does not start because the store-and-forward function is active.

For information on store-and-forward processing, see How the store-and-forward feature works.

Event sequencing in a network deployment environment:

Event sequencing can be used in a network deployment environment, with or without a high availability manager. Consult the table in this topic to ensure that your particular topology is supported.

Note that Service Component Architecture (SCA) destinations for any component using event sequencing cannot be partitioned. Therefore, if you are using clusters, you can have only one active messaging engine per cluster.

Table 50. Event sequencing support in a network deployment environment

Topology	Is event sequencing supported?
Standalone server	Yes
No clusters	Yes
Applications are clustered. Messaging engines and destinations are not clustered.	Yes
Messaging engines are clustered. Applications and destinations are not clustered.	Yes
Messaging engines and destinations are clustered. Applications are not clustered.	No. Clustered destinations are partitioned and cannot be used with event sequencing.
Applications and messaging engines are clustered (same cluster). Destinations are not clustered.	Yes
Applications, messaging engines, and destinations are clustered (same cluster).	No. Clustered destinations are partitioned and cannot be used with event sequencing.
Applications and messaging engines are clustered (different clusters). Destinations are not clustered.	Yes
Applications, messaging engines, and destinations are clustered (different clusters).	No. Clustered destinations are partitioned and cannot be used with event sequencing.

Using event sequencing in a high availability environment

High availability (HA) support means that system subcomponents, such as the event sequencing runtime, are made highly available and the workload can be distributed in the case of a node or daemon failure.

Although event sequencing requires a singleton service to process the event messages on a destination, an HA manager provides the necessary services to ensure that this process is not a single point of failure. Instead, the event sequencing runtime fails over to another server in the cluster in the event of a system lock up.

Failed sequenced events:

Processing errors or unavailable resources can cause a sequenced event to fail. How any remaining events in the sequence are handled is determined by the setting of the “Process requests when error encountered” attribute of the event sequencing qualifier in IBM Integration Designer.

The “Process requests when error encountered” attribute of the event sequencing qualifier has two possible values.

Checked
Use this value if you want the processing of the sequence of events to ignore the failure and proceed to process the next event in the sequence.

Unchecked
Use this value if you want to halt the processing of dependent events in the sequence until the failure is resolved. You can use the failed event manager to quickly identify failed sequenced events and resubmit them for processing.

When this attribute is unchecked and a sequenced event is unsuccessfully processed and sent to the failed event manager, you can handle it in one of the following ways:

- resubmit it without modification
- resubmit it with modification (either with or without changing event sequencing key identifiers)
- delete it (the Recovery subsystem uses the event sequencing callback to delete the lock associated with the deleted event to allow remaining events in the sequence to be processed)

If the resubmission is successful, the event is processed in its original sequenced position within the queue.

Limitations in event sequencing:

Certain types of components and invocations offer limited support for event sequencing.

Limitations for the current release of event sequencing include the following:

- To effectively use event sequencing on a Service Component Architecture (SCA) component that has asynchronous invocations, it is recommended that you use the request-response method signature. The event sequencing runtime interprets a response as a signal that the work is complete and releases the lock.

  Note: If you cannot declare a method as a request-response operation, you might need to specify event sequencing on downstream components, making sure you use the same event sequencing key for all methods.

Event sequencing is not supported in the following scenarios:

- Using unmanaged threads or non-SCA bindings to send events to their destinations without proper session context.
- Using synchronous invocations to components with asynchronous invocations.

Enabling event sequencing in WebSphere ESB

Event sequencing provides the ability to sequence incoming events to an SCA component in WebSphere ESB. Export bindings are the entry points to the target SCA components. For event sequencing to be enabled, the exports must process and deliver messages in the same order in which those messages are received.

Enabling event sequencing: EIS exports:

WebSphere adapters provide a mechanism to allow event sequencing in WebSphere ESB by specifying an activation specification property. Also, the export must process and deliver messages in the same order it received those messages.

About this task

For more details about the activation specification property; see the WebSphere adapter documentation. For JCA 1.5 resource adapters, consult the specific provider documentation for details on how to configure the adapter to enable the ordering or sequencing of events.

In general, if event sequencing is required in a network deployment environment, the module that has the export should be moved to a standalone server or to a cluster that has only one active server that is enabled for high availability.

Enabling event sequencing: JMS exports:

To enable event sequencing for JMS export bindings in WebSphere ESB, you must configure properties of the binding. Also, the export must process and deliver messages in the same order that it receives those messages.

Before you begin

Event sequencing for JMS export bindings is supported in a clustered environment only when the destinations are not partitioned. For event sequencing to function in a network deployment environment with clusters, there can be only one active messaging engine per cluster. For event sequencing in a network deployment environment with servers that are not in a cluster, each server can have an active messaging engine.

About this task

Event sequencing requires events to acquire a lock before being dispatched to the target component for processing. When processing is complete, the event releases the lock. If an event cannot acquire a lock, processing of the invocation is suspended. If the event subsequently acquires a lock, it will be dispatched.

You declare that event sequencing is required on a particular method for a particular component by adding an event sequencing qualifier to the method in the component definition.

- The keySpecification attribute defines the key that will be used to identify the events that need to be sequenced.
- The parameter attribute specifies the parameter from which the key attributes will be extracted.
- The name attribute is the name of the parameter.
- The xpath attribute is applied to the parameter to extract a value that will be part of the key.

You must specify a parameter element for each parameter that is going to contribute to the key.

The **esadmin** command line utility can be used to list locks and delete locks, both active and queued.

You enable event sequencing for a JMS export from Integration Designer.

Procedure

1. In Integration Designer, click the **Properties** tab for the export.
2. From the **Message configuration** section, select the **Event sequence required** check box.

Results

Event sequencing is enabled for your binding.

Note: Removing the exception destination means that any failure will stop all incoming messages.

Enabling event sequencing: Generic JMS exports:

To enable event sequencing for Generic JMS export bindings in WebSphere ESB, you must configure properties of the binding. Also, the export must process and deliver messages in the same order that it receives those messages.

Before you begin

If event sequencing for Generic JMS export bindings is required in a network deployment environment, the module that has the export should be moved to a standalone server or to a cluster that has only one active server that is enabled for high availability.

About this task

You enable event sequencing for a Generic JMS export from Integration Designer.

Procedure

1. In Integration Designer, click the **Properties** tab for the export.
2. From the **Message configuration** section, select the **Event sequence required** check box.

Enabling event sequencing: WebSphere MQ JMS exports:

To enable event sequencing for WebSphere MQ JMS export bindings in WebSphere ESB, you must configure properties of the binding. Also, the export must process and deliver messages in the same order that it receives those messages.

Before you begin

If event sequencing for these export bindings is required in a network deployment environment, the module that has the export should be moved to a standalone server or to a cluster that has only one active server that is enabled for high availability.

About this task

You enable event sequencing for WebSphere MQ JMS export bindings from Integration Designer.

Procedure

1. In Integration Designer, click the **Properties** tab for the export.
2. From the **Message configuration** section, select the **Event sequence required** check box.

Enabling event sequencing: WebSphere MQ exports:

To enable event sequencing for WebSphere MQ export bindings in WebSphere ESB, you must configure properties of the binding. Also, the export must process and deliver messages in the same order that it receives those messages.

Before you begin

If event sequencing for WebSphere MQ export bindings is required in a network deployment environment, the module that has the export should be moved to a standalone server or to a cluster that has only one active server that is enabled for high availability.

About this task

You enable event sequencing for a WebSphere MQ export from Integration Designer.

Procedure

1. In Integration Designer, click the **Properties** tab for the export.
2. From the **Message configuration** section, select the **Event sequence required** check box.

Listing, releasing, and deleting locks

The lock manager handles event sequencing locks. You can use the **esAdmin** command to list, delete, or unlock any lock in the lock manager.

The lock manager supports two operations on event locks:

- Lock: The lock operation attempts to acquire a lock and stores the lock request in a database. After a lock is granted, processing resumes for the invocation that requested the lock.
- Unlock: The unlock operation releases the current lock and grants the lock to the next lock request.

Requests for the same lock are put into a queue in the order in which they are received. Locks are persisted to the default WebSphere ESB database and data source to ensure they can be recovered in the case of a server failure.

The **esAdmin** command enables you to administer the active and queued locks currently in the lock manager. The following sections provide more detail on using **esAdmin**.

Note: If you are using partitioned databases, run the **esAdmin** command once for each deployment target. In a clustered environment, you can run it on any cluster, but do not run it on the deployment manager.

Listing locks

The **esAdmin** command can list all active and queued locks in the lock manager, or only those locks associated with a specific module, component, or method.

Use one of the following methods with **esAdmin**:

- listAll: Lists all active and queued locks in the lock manager.
- listLocks: Lists a subset of the active and queued locks in the lock manager. Specify one or more of the following parameters to return a filtered list of locks:
  - moduleName
  - componentName
  - methodName

  For example, the following command returns a list of active and queued locks for the CustComp component that is part of the CusMod module.

  ```
  esAdmin listLocks CustMod CustComp
  ```

The command returns output that looks like the following:

Table 51. Sample output from esAdmin listLocks command

Lock Id	Sequence Id	Owner Id	Module	Component	Method	System Message Id
7564504	2	695376	CustMod	CustComp	createCust	A09-427BE_5002
7564504	3	232757	CustMod	CustComp	createCust	ADF-053RT_5004

In the output above, the sequence ID is the order in which the lock requests are queued; the lowest number in the sequence currently holds the lock. The system message ID specifies the ID for the corresponding service integration bus message; you can use this information to correlate lock requests with the messages on the destinations.

Releasing locks

Use the **esAdmin** command to release a single lock, as follows:

`esAdmin unlock lockId.`

lockId is the unique lock ID returned by the `esAdmin listLock` or `esAdmin listAll` command.

This command is useful when you encounter a deadlock; you can release the lock that is deadlocked and grant it to the next lock request in the queue.

Deleting locks

If you need to delete one or more locks, first stop the module associated with the lock. Then, use the esAdmin command to delete the lock from the database.

For examples:

`esAdmin deleteLocks moduleName`

You must restart the module in order for the destinations to resume processing event messages.

Use the esAdmin deleteLocks command with caution. All locks in the specified module are deleted from the lock manager database.

Troubleshooting event sequencing

Refer to the information in this topic if you are experiencing difficulty with event sequencing.

Problems with the event sequencing qualifier

Ensure that your component definition is correct:

- Is the event sequencing qualifier set on the method? Event sequencing validation fails if the qualifier is erroneously set on the interface.
- Is the parameter name valid?
- Is the xpath element valid, and does it correctly resolve to a primitive?
- Is there a single eventSequencing element for the method? Each method supports only one eventSequencing element.
- Is there a single keySpecification element for the method? Each method supports only one keySpecification element.

Deadlocks

Deadlocks occur when an invoked operation with a lock invokes another operation on the same component using the same event sequencing key and group. You can resolve a deadlock by using the **esAdmin** command to list and release the current lock.

To avoid deadlocks, carefully consider dependencies when implementing event sequencing. Ensure that operations with circular dependencies are in different event sequencing groups.

Event sequencing callback fails to release a lock

While trying to delete a failed sequenced event in the Recovery subsystem, the event sequencing callback can fail to release the event's lock. This typically occurs when a target application has been removed or when other components of the system (for example, the database) are unavailable.

In this situation, the failed event manager generates an error message. Use the **esAdmin** command to manually delete the lock associated with the failed event.

Performance issues

If you are experiencing memory problems on the messaging engine server used for event sequencing components, try modifying the runtime event sequencing properties in the *install_root*/properties/eventsequencing.properties file.

The maxActiveMessages property defines the number of messages currently locked on a component destination; too many large messages can negatively affect performance and cause memory problems. Note that a value of 0 (zero) means that an unlimited number of messages are allowed. By default, the maxActiveMessages property is set to 100. When changing the value, consider using the following formula where *delta* is the standard deviation of the accuracy of the estimate for the anticipated number of sequenced events with the same sequencing key that can be simultaneously processed.

```
average_number_of_ES_keys * average_number_of_potential_queued_events_per_key
+ delta
```

The workItemRetryCount property sets the upper boundary for the verification work retry count. A verification work item is spawned when an asynchronous event is unlocked and there are dependent events waiting to be processed. In this situation the creation and deletion of the lock are done in separate units of work and the work verification task ensures that the processing of one unit of work is complete before the next event is processed. By default, workItemRetryCount is set to -1 (retry).

The workItemSleepTime property specifies the amount of time that elapses between work verification retry attempts. By default, workItemSleepTime is set to 10 seconds. Note that lowering the value can decrease performance.

To modify any of the properties, perform the following steps.

1. Open the eventsequencing.properties file in a text editor.
2. Make the appropriate modifications for your environment.
3. Save and close the file.

4. Stop and restart any applications that are part of the event sequencing component in order for the changes to take effect.

Managing failed events in WebSphere ESB

When an event fails in WebSphere ESB, it is stored in a database in the failed events manager. You can search for a failed event, and then view, modify, resubmit, or delete it.

To view, modify, resubmit, or delete any failed event, as described in the following sections, the first step is to click **Integration Applications** > **Failed Event Manager**.

Failed SCA events

A Service Component Architecture (SCA) event is a request or response that is received by a service application. It might come from an external source, such as an inbound application adapter, or an external invocation to a web service. The event consists of a reference to the business logic that it wants to operate and its data, which is stored in a Service Data Object. When an event is received, it is processed by the appropriate application business logic.

A single thread of execution might branch off into multiple branches. The individual branches are linked to the main invoking event by the same session.

If the business logic in a branch cannot run due to system failure, component failure, or component unavailability, a failed event is created. If multiple branches fail, a failed event is created for each branch. The Recovery service handles the following types of failed SCA events:

- Event failures that occur during an asynchronous invocation of an SCA operation
- Event failures that are caused by a runtime exception, for example, any exception that is not declared in the methods used by the business logic

The Recovery service does not handle failures from synchronous invocations.

Failed SCA events typically have source and destination information associated with them. The source and destination are based on the failure point (the location where the invocation fails), regardless of the type of interaction. Consider the following example, where Component A is asynchronously invoking Component B. The request message is sent from A to B, and the response callback message is sent from B to A.

- If the exception occurs during the initial request, Component A is the source and Component B is the destination for the failed event manager.
- If the exception occurs during the response, Component B is the source and Component A is the destination for the failed event manager.

This pattern is true for all asynchronous invocations.

The Recovery service sends failed SCA asynchronous interactions to failed event destinations that have been created on the SCA system bus (SCA.SYSTEM.*cell_name*.Bus). The data for failed events is stored in the failed event database (by default, WPCRSDB). You can find the data in the failed event manager.

Failed WebSphere MQ events

A WebSphere MQ event might fail if there is a problem such as a data-handling exception in the WebSphere MQ binding export or import used by an SCA module.

You can use the recovery binding property to enable or disable recovery for each WebSphere MQ binding at authoring time. You can set the `recoveryMode` property to one of the following settings:

`bindingManaged`	Allow binding to manage recovery for failed messages
`unmanaged`	Rely on transport-specific recovery for failed messages

Recovery for WebSphere MQ bindings is enabled by default. When it is enabled, WebSphere MQ failed events are created in the following situations:

- The function selector fails
- The fault selector fails
- The fault selector returns the `RuntimeException` fault type
- The fault handler fails
- The data binding or data handler fails after another attempt in WebSphere MQ

In addition, a failed SCA event is created when the `ServiceRuntimeException` exception is thrown in a WebSphere MQ binding target component after another attempt in WebSphere MQ.

These failures might occur during inbound or outbound communication. During outbound communication, MQImport sends a request message and receives the response message. A failed event is generated if the WebSphere MQ import binding detects a problem while it is processing the service response. During inbound communication, the following sequence of events occurs:

1. MQExport receives the request message.
2. MQExport starts the SCA component.
3. The SCA component returns a response to MQExport.
4. MQExport sends a response message.

A failed event is generated if the WebSphere MQ export binding detects a problem while it is processing the service request.

The Recovery service captures the WebSphere MQ message and stores it in the failed event database. It also captures and stores the module name, component name, operation name, failure time, exception detail, and WebSphere MQ properties of the failed event. You can use the failed event manager to manage failed WebSphere MQ events, or you can use a custom program.

To disable recovery, set the `recoveryMode` property to `unmanaged`.

Important: If the `recoveryMode` property is missing for earlier versions of applications, the recovery capability is regarded as enabled.
When recovery is disabled, a failed message is rolled back to its original destination and tried again. The system does not create a failed event.

Failed JMS events

The Java Message Service (JMS) binding type and configuration determine whether a failed event is generated and sent to the failed event manager.

You can use the recovery binding property to enable or disable recovery for each JMS binding at authoring time. You can set the `recoveryMode` property to one of the following settings:

`bindingManaged`	Allow binding to manage recovery for failed messages
`unmanaged`	Rely on transport-specific recovery for failed messages

Recovery for JMS bindings is enabled by default. When it is enabled, JMS failed events are created in the following situations:

- The function selector fails
- The fault selector fails
- The fault selector returns the `RuntimeException` fault type
- The fault handler fails
- The data binding or data handler fails after one repetition in JMS

In addition, a failed SCA event is created when the `ServiceRuntimeException` exception is thrown in a JMS binding target component after another attempt in JMS.

These failures might occur during inbound or outbound communication. During outbound communication, JMSImport sends a request message and receives the response message. A failed event is generated if the JMS import binding detects a problem while it is processing the service response. During inbound communication, the following sequence of events occurs:

1. JMSExport receives the request message.
2. JMSExport starts the SCA component.
3. The SCA component returns a response to JMSExport.
4. JMSExport sends a response message.

A failed event is generated if the JMS export binding detects a problem while it is processing the service request.

The Recovery service captures the JMS message and stores it in a Recovery table in the Common database. It also captures and stores the module name, component name, operation name, failure time, exception detail, and JMS properties of the failed event. You can use the failed event manager to manage failed JMS events, or you can use a custom program.

To disable recovery, set the `recoveryMode` property to `unmanaged`.

Important: If the `recoveryMode` property is missing for earlier versions of applications, the recovery capability is regarded as enabled.
When recovery is disabled, a failed message is rolled back to its original destination and tried again. The system does not create a failed event.

Tip: For service runtime exceptions that are generated when a requested service is unavailable, you can use the Store and Forward feature to prevent further failures.

You specify a store-and-forward qualifier when you configure a component to be started asynchronously. When a runtime error is generated by that component, subsequent events (in this case, asynchronous requests) are prevented from reaching the component. For more information, see “Preventing failures when a service is unavailable”.

Related concepts:

“Preventing multiple failures when a service is unavailable” on page 582
If your component calls a service, and that service is unavailable, a runtime error occurs. Events associated with the error are treated, by default, as failed events. Using the store-and-forward feature, however, you can drastically reduce the number of failed events that are generated by the system.

Security considerations for recovery

If you have enabled security for your WebSphere ESB applications and environment, it is important to understand how role-based access and user identity affect the Recovery subsystem.

Role-based access for the failed event manager
: The failed event manager uses role-based access control for the failed event data and tasks. Only the administrator and operator roles are authorized to perform tasks within the failed event manager. Users logged in as either administrator or operator can view all data associated with failed events and can perform all tasks.

Event identity and user permissions
: A failed event encapsulates information about the user who originated the request. If a failed event is resubmitted, its identity information is updated to reflect the user who resubmitted the event. Because different users logged in as administrator or operator can resubmit events, these users must be given permissions to the downstream components required to process the event.

For more information about implementing security, see Securing applications and their environment.

Finding failed events in WebSphere ESB

You can use the failed event manager to find failed events in WebSphere ESB. You can search for all failed events, or for a specific subset of events, on all the servers within the cell.

Before you begin

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

About this task

This topic describes how to find all failed events in a cell. This default query returns all failed events for Service Component Architecture (SCA) and Java Message Service (JMS). If Business Process Choreographer is installed, the query also returns failed, terminated, and stopped Business Process Choreographer events. For more information about different types of failed events, see "Managing failed events in WebSphere ESB".

Procedure

To retrieve a complete list of failed events, complete the following steps:

1. Ensure that the administrative console is running.
2. Click **Integration Applications** > **Failed Event Manager**.
3. In the **Failed events on this server** box, click **Get all failed events**.

Results

The Search Results page opens with a list of all the WebSphere ESB failed events in the cell.

What to do next

You can now view (and in some cases, modify) data in a failed event, resubmit it, or delete it.

Searching for events by criteria:

Use the failed event manager Search page to find only those events that meet specified criteria. You can search by failed event type and by criteria such as failure time, event destination or source, exception or business object type, session ID or event sequencing qualifier.

Before you begin

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

About this task

To search for a specific subset of failed events on the server, perform the following steps.

Procedure

1. Ensure the administrative console is running.
2. Click **Integration Applications** > **Failed Event Manager** to enter the failed event manager.
3. From the **Failed events on this server** box, click **Search failed events**.
4. From the **Event type** box on the Search failed events page, select one or more types of events to search for:
   - SCA
   - JMS
   - WebSphere MQ
5. Optional: Specify any additional search criteria. The following table describes the available options. If you specify multiple criteria, the AND operator is used during the query; the failed event manager returns only events that meet all of the criteria.

Table 52. Search criteria

Search criteria	Field or fields to use	Supported event types	Usage notes
The module, component, or method the event was en route to when it failed.	**Module** **Component** **Operation**	SCA JMS WebSphere MQ	Use one or more of these fields to search for failed events associated with a specific module, component, or method.
The time period during which the event failed	**From date** **To date**	SCA JMS WebSphere MQ	Formats for date and time are locale-specific. An example is provided with each field. If the value you provide is not formatted correctly, the failed event manager displays a warning and substitutes the default value for that field. The time is always local to the server. It is not updated to reflect the local time of the individual workstations running the administrative console.
The session in which the event failed	**Session ID**	SCA	None
The module or component from which the event originated	**Source module** **Source component**	SCA	Use one or both of these fields to find only those failed events that originated from a specific source module or component. The failed event manager determines the source based on the point of failure, regardless of interaction type.
The type of business object in the failed event	**Business object type**	SCA	None
Whether the event had the event sequencing qualifier specified	**Event sequencing qualifier**	SCA	None
Whether the event caused the store to be started	**Store and forward qualifier**	SCA	None
Whether the event was caused because a failure response could not be sent to Business Process Choreographer	**Process response qualifier**	SCA	None
The exception thrown when the event failed	**Exception text**	SCA	Specify all or part of the exception text in the field to find all events associated with that exception.

For detailed information about each field and the values it accepts, see the online help for the failed event manager Search page.

6. Click **OK** to begin the search.

What to do next

You can now view (and in some cases, modify) data in a failed event, resubmit it, or delete it.

Working with data in failed events

Each failed event has data associated with it; often, that data can be edited before an event is resubmitted. There are two basic types of data for a failed event: data about the event, and business data.

Data about the failed event

All failed events have the following data:

- The event ID, type, and status
- The time the event failed
- The deployment target associated with the event

In addition, SCA, JMS, WebSphere MQ, Business Process Choreographer, and Business Flow Manager hold queue events have data specific to the event type:

- SCA events
  - The session ID
  - The type of service invocation used between SCA components
  - The names of the module and component from which the event originated (the source).
  - The names of the destination module, component and method for the event
  - Whether an event sequencing qualifier has been declared for this event
  - The destination module where the event has been or will be resubmitted
  - The correlation ID, if one exists
  - The exception thrown when the event failed
  - The expiration date for resubmitted events (this data can be edited)
  - The trace control set for the event (this data can be edited)
- JMS events:
  - The type of service invocation used
  - The names of the destination module, component and method for the event
  - The exception thrown when the event failed
  - The destination module where the event has been or will be resubmitted
  - The correlation ID, if one exists
  - The expiration date for resubmitted events (this data can be edited)
  - The JMS-specific properties associated with the failed event:
    - The message type and priority
    - The JMS destination
    - The delivery mode
    - Redelivery data, including the redelivered count and redelivered indicator (`true` or `false`)
    - The destination replies are sent to for request-response or two-way interactions
- WebSphere MQ events:
  - The type of service invocation used
  - The names of the destination module, component and method for the event

- The exception thrown when the event failed
- The destination module where the event has been or will be resubmitted
- The correlation ID, if one exists
- The expiration date for resubmitted events (this data can be edited)
- The WebSphere MQ-specific properties associated with the failed event:
  - The message type, format, and priority
  - The WebSphere MQ destination
  - The delivery mode
  - Redelivery data, including the redelivered count and redelivered indicator (`true` or `false`)
  - The reply-to queue and queue manager

Business data

SCA failed events typically include business data. Business data can be encapsulated in a business object, or it can be simple data that is not part of a business object. Business data for SCA failed events can be edited with the business data editor available in the failed event manager.

Browsing data in failed events:

Use the failed event manager to view failed event data and any business data associated with the event.

Before you begin

If administrative security is enabled, you must be logged as administrator or operator to perform this task.

About this task

To browse failed event data, use the following procedure.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the Search Results page of the failed event manager, click the ID (found in the Event ID column) of the failed event whose data you want to browse.

   The Failed Event Details page opens and displays all of the information about the event.
3. If your failed event has business data, you can browse it by clicking **Edit business data**.

   The Business Data Editor collection page opens, displaying the business data associated with the failed event. Each parameter name in the hierarchy is a link. If the parameter is a simple data type, clicking its name will open up a form so you can edit the parameter's value. If the parameter is a complex data type, clicking its name will expand the hierarchy further.

Editing trace or expiration data in a failed SCA event:

The Failed Event Details page enables you to set or modify values for the trace control and expiration date associated with a failed event.

Before you begin

You must be logged in as administrator or operator to perform this task.

About this task

Important: Any edits you make to the trace or expiration data are only saved locally until you resubmit the event. If you perform any other action before resubmitting the event, all edits are lost.

Failed Service Component Architecture (SCA) events can be resubmitted with trace to help you monitor the event processing. Tracing can be set for a service or a component, and it can be sent to a log or to the Common Event Infrastructure (CEI) server. When you view the failed event data on the Failed Event Details page, the default trace value `SCA.LOG.INFO;COMP.LOG.INFO` is shown for the event. If you resubmit the event with this default setting, no trace occurs when the session calls an SCA service or executes a component.

Some failed SCA events also have an expiration. If a user has specified an expiration with the asynchronous call that sends the event, that data persists even if the event fails, and the expiration time appears in the **Resubmit Expiration Time** field on the Failed Event Details page. Expired failed events cannot be resubmitted successfully. To prevent a second failure, you can edit the expiration date for the event to ensure that it is not expired when it is resubmitted.

To edit trace or expiration data in a failed event, use the following procedure.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the failed event manager's Search Results page, click the ID (found in the Event ID column) of the failed event whose data you want to edit.

   The Failed Event Details page opens.
3. If the event has an expiration date that causes it to expire before it is resubmitted, edit the expiration in the **Resubmit expiration time** field.

   The expiration time shown is local to the server. The value for this field must be formatted according to your specified locale. An example of the correct format for your locale is provided above the field.
4. If you want to enable tracing for the failed event, specify a new value in the **Trace Control** field. For detailed information about trace values, see the Monitoring topics in the WebSphere Business Process Management Information Center.
5. Do one of the following:
   - If the edited data is correct and you want to resubmit the event, click **Resubmit** to make the changes at a server level.
   - If you want to remove the changes you made, click **Undo local changes**.

   The edited failed event is resubmitted for processing and is removed from the failed event manager.

Related tasks:

"Finding failed events in WebSphere ESB" on page 566
You can use the failed event manager to find failed events in WebSphere ESB. You can search for all failed events, or for a specific subset of events, on all the servers within the cell.

Editing business data in a failed SCA event:

Business data can be encapsulated into a business object, or it can be simple data that is not part of a business object. A failed event can have both simple data and a business object associated with it. Use the business data editor to edit the business data associated with a failed event before you resubmit it.

Before you begin

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

About this task

For each failed event, the editor displays the associated business data in a hierarchical format; the navigation tree is updated as you navigate through the parameters to give you a clear picture of where you are in the hierarchy.

You can edit only simple data types (for example, String, Long, Integer, Date, Boolean). If a data type is complex (for example, an array or a business object), you must navigate through the business data hierarchy until you reach the simple data types that make up the array or business object. Complex data is denoted by an ellipsis (**...**) in the Parameter Value column.

Important: Any edits you make to business data are saved locally. Changes are not made to the corresponding business data in the server until you resubmit the failed event.

To edit business data associated with a failed Service Component Architecture (SCA) event, use the following procedure.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the failed event manager's Search Results page, click the ID (found in the Event ID column) of the failed event that has data you want to edit.
3. From the failed event detail page, click **Edit business data** to access the Business Data Editor collection page.

   This page displays a hierarchical view of all the data associated with the failed event.
4. Navigate through the business data hierarchy by clicking the name of each parameter (these appear as links in the Parameter Name column). When you have located the parameter with values you want to edit, click its name.

   If the parameter has an editable value, the Business Data Editor page opens.
5. In the **Parameter value** field, specify the new value for the parameter.
6. Click **OK**.

   The change is saved locally and you are returned to the Business Data Editor collection page.

7. If you want to remove the changes you made, click **Undo local business data changes**.

   All the edits are removed and the business data is returned to its original state.
8. If the edited business data is correct, click **Resubmit** to make changes at a server level.

   The edited failed event is resubmitted for processing and is removed from the failed event manager.

Resubmitting failed events in WebSphere ESB

You can resubmit a failed event in WebSphere ESB from the failed event manager. You can resubmit an event without changes, or, in some cases, you can edit the business data parameters before you resubmit it.

When a failed event is resubmitted, the processing resumes only for the failed branch and not for the entire event.

Important: If one of the following conditions occurs while an application is running in a unit-test environment or a production environment, you cannot resubmit the failed event:

- The application is stopped on the server
- The input and output parameters are modified
- The schema for the input and output parameters is modified

You can trace resubmitted SCA events to monitor the processing of the event. You can set tracing for a service or a component, and you can send the output to a log or to the Common Event Infrastructure (CEI) server.

You can also use the unique ID of the event to track its success or failure. If a resubmitted event fails again, it is returned to the failed event manager with its original event ID and an updated failure time.

Resubmitting an unchanged failed event:

You can resubmit one or more unchanged failed events to be processed again. Processing resumes only for the failed branch, not for the entire event.

About this task

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the Search Results page, select the check box next to each failed event you want to resubmit.
3. Click **Resubmit**.

Results

Each selected event is resubmitted for processing and is removed from the failed event manager.

Resubmitting a failed SCA event with trace:

You can monitor the resubmission of a failed Service Component Architecture (SCA) event to determine whether it now succeeds. The failed event manager provides optional tracing for all failed events.

About this task

Tracing can be set for a service or a component, and it can be output to a log or to the Common Event Infrastructure (CEI) server. For detailed information about setting and viewing trace, see Monitoring.

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the Search Results page, select the check box next to each failed event you want to resubmit.
3. Click **Resubmit with trace**.
4. From the Resubmit with Trace page, specify the level of trace you want to use in the **Trace control** field.

   By default, the value is SCA.LOG.INFO;COMP.LOG.INFO. With this setting, no trace occurs when the session calls an SCA service or executes a component.
5. Click **OK** to resubmit the failed event and return to the Search Results page.

What to do next

To view the trace log for a resubmitted event, open the corresponding component logger or use the CEI log viewer.

Resubmitting failed Business Process Choreographer responses:

When a failure response cannot be delivered to the requesting business process due to an infrastructure problem, an event is stored in the failed event database. These type of events will have a process response qualifier specified on them. You can resubmit these failed events to either the request queue or the response queue using the failed event manager.

About this task

To resubmit a failed SCA event, perform the following steps.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the Search Results page, select the check box next to each failed event you want to resubmit.
3. Click **Resubmit** or **Resubmit with trace**.
4. If the Process Response event qualifier is defined for the failed event, a resubmit page will appear. Select **Resubmit requests to the destination** or **Resubmit the exception response to the source**. Selecting **Resubmit the exception response to the source** allows the event to be sent to the response queue without having to be reprocessed.

Results

Depending on whether you selected **Resubmit requests to the destination** or **Resubmit the exception response to the source**, the event will be resubmitted to the appropriate queue.

Managing failed SCA events

When problems processing a Service Component Architecture (SCA) request or response message create a failed SCA event in the Recovery subsystem, you must decide how to manage that event. Use the information in this topic to help you identify and fix the error and clear the event from the Recovery subsystem.

About this task

Failed SCA events typically have source and destination information associated with them. The source and destination are based on the failure point (the location where the invocation fails), regardless of the type of interaction. Because runtime exceptions are not declared as part of the interface, component developers should attempt to resolve the exception and thus prevent a runtime exception from inadvertently being propagated to the client if the client is a user interface.

To manage a failed SCA event, perform the following steps.

Procedure

1. Use the failed event manager to locate information about the failed SCA event, taking note of the exception type.
2. Locate the exception type in Table 53 to determine the location and possible causes of the error, as well as suggested actions for managing the failed event.

Table 53. Failed SCA events

Exception type	Possible cause of error	Suggested action
ServiceBusinessException	A business exception occurred during the execution of a business operation.	Look at the exception text to determine the exact cause, and then take appropriate action.
ServiceExpirationRuntimeException	A SCA asynchronous message has expired.	Set the expiration time using the RequestExpiration qualifier on the service reference. Investigate why the service is not responding fast enough.
ServiceRuntimeException	A runtime exception occurred during the invocation or execution of a service.	Look at the exception text to determine the exact cause, and then take appropriate action.
ServiceTimeoutRuntimeException	Response to an asynchronous request was not received within the configured period of time.	Set the expiration time using the RequestExpiration qualifier on the service reference. Investigate why the service is not responding fast enough.
ServiceUnavailableException	This exception is used to indicate that there was an exception thrown while invoking an external service via an import.	Look at the exception text to determine the exact cause, and then take appropriate action.
ServiceUnwiredReferenceRuntimeException	A SCA reference used to invoke a service is not wired correctly.	Look at the exception text to determine the exact cause, and then take appropriate action to correctly wire the SCA reference.

Managing failed JMS events

When problems processing a JMS request or response message create a failed JMS event in the Recovery subsystem, you must decide how to manage that event. Use the information in this topic to help you identify and fix the error and clear the event from the Recovery subsystem.

About this task

To manage a failed JMS event, perform the following steps.

Procedure

1. Use the failed event manager to locate information about the failed JMS event, taking note of the exception type.
2. Locate the exception type in Table 54 to determine the location and possible causes of the error, as well as suggested actions for managing the failed event.

Table 54. Failed JMS events

Exception type	Location of error	Possible cause of error	Suggested action
FaultServiceException	Fault handler or fault selector	There is malformed data in the JMS message.	1. Inspect the JMS message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the fault handler or fault selector.	1. Debug the custom fault selector or fault handler, fixing any errors identified. 2. Resubmit the failed event.
ServiceRuntimeException	Fault handler	The fault selector and runtime exception handler are configured to interpret the JMS message as a runtime exception. This is an expected exception.	Look at the exception text to determine the exact cause, and then take appropriate action.

Table 54. Failed JMS events (continued)

Exception type	Location of error	Possible cause of error	Suggested action
DataBindingException or DataHandlerException	Data binding or data handler	There is malformed data in the JMS message.	1. Inspect the JMS message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the data binding or data handler.	1. Debug the custom data binding or data handler, fixing any errors identified. 2. Resend the message. 3. Delete the failed event.
SelectorException	Function selector	There is malformed data in the JMS message.	1. Inspect the JMS message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the function selector.	1. Debug the custom function selector, fixing any errors identified. 2. Resend the message. 3. Delete the failed event.

Managing failed WebSphere MQ events

When problems processing a WebSphere MQ request or response message create a failed WebSphere MQ event in the Recovery subsystem, you must decide how to manage that event. Use the information in this topic to help you identify and fix the error and clear the event from the Recovery subsystem.

About this task

To manage a failed WebSphere MQ event, perform the following steps.

Procedure

1. Use the failed event manager to locate information about the failed event, taking note of the exception type.
2. Locate the exception type in Table 55 on page 578 to determine the location and possible causes of the error, as well as suggested actions for managing the failed event.

Table 55. Failed WebSphere MQ events

Exception type	Location of error	Possible cause of error	Suggested action
FaultServiceException	Fault handler or fault selector	There is malformed data in the WebSphere MQ message.	1. Inspect the message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the fault handler or fault selector.	1. Debug the custom fault selector or fault handler, fixing any errors identified. 2. Resubmit the failed event.
ServiceRuntimeException	Fault handler	The fault selector and runtime exception handler are configured to interpret the WebSphere MQ message as a runtime exception. This is an expected exception.	Look at the exception text to determine the exact cause, and then take appropriate action.
DataBindingException or DataHandlerException	Data binding or data handler	There is malformed data in the WebSphere MQ message.	1. Inspect the message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the data binding or data handler.	1. Debug the custom data binding or data handler, fixing any errors identified. 2. Resend the message. 3. Delete the failed event.

Table 55. Failed WebSphere MQ events (continued)

Exception type	Location of error	Possible cause of error	Suggested action
SelectorException	Function selector	There is malformed data in the WebSphere MQ message.	1. Inspect the message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the function selector.	1. Debug the custom function selector, fixing any errors identified. 2. Resend the message. 3. Delete the failed event.

Finding Common Base Events related to a failed event

A failed event can be related to one or more Common Base Events. The failed event manager provides a link to view related Common Base Events in the Common Base Event Browser.

Before you begin

You must be logged in as administrator or operator to perform this task.

About this task

Examining related Common Base Events can give you additional information about how or why the original event failed. The failed event and any related Common Base Events are linked by the same session ID.

To find and view related Common Base Events, use the following procedure.

Procedure

1. From within the administrative console, use the failed event manager to locate the failed event you want to investigate. See "Finding failed events in WebSphere ESB" on page 566 for instructions on how to search for failed events.
2. From the Failed Event Details page for that event, click **Browse Related Common Base Events**.

Results

The Common Base Event Browser opens in a new browser window and lists any Common Base Events related to the original failed event.

Deleting failed events

If you do not want to resubmit a failed event, or if you have failed events that have expired, use the failed event manager to delete them from the server. The failed event manager provides three options for deleting failed events.

Before you begin

You must be logged in as administrator or operator to perform this task.

About this task

To delete one or more failed events, use the following procedure.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the failed event manager's Search Results page, do one of the following:
   - If you want to delete one or more specific failed events, select the check box next to each event and then click **Delete**.
   - If you want to delete only those failed events that have expired, click **Delete expired events**. Note that this deletes only the expired events in the current set of search results.
   - If you want to delete all failed events on the server, click **Clear all**.

Troubleshooting the failed event manager

This topic discusses problems that you can encounter while using the failed event manager.

Select the problem you are experiencing from the table below:

Problem	Refer to the following
I am having trouble entering values in the Search page's **By Date** tab	"Values in the By Date and From Date field automatically change to default if entered incorrectly"
I am having trouble deleting expired events	"Using the Delete Expired Events function appears to suspend the failed event manager" on page 581
I am having trouble with failed events not being created	"Failed events are not being created" on page 581
I am having trouble retrieving or deleting large numbers of failed events	"The server fails when retrieving or deleting large numbers of failed events" on page 581

Values in the By Date and From Date field automatically change to default if entered incorrectly

The Search page's **From Date** and **To Date** fields require correctly formatted locale-dependent values. Any inconsistency in the value's format (for example, including four digits in the year instead of 2, or omitting the time) will cause the failed event manager to issue the following warning and substitute a default value in the field:

```
CWMAN0017E: The date entered could not be parsed correctly:
your_incorrectly_formatted_date. Date: default_date is being used.
```

The default value of the **From Date** field is defined as `January 1, 1970, 00:00:00 GMT`.

Important: The actual default value shown in your failed event manager implementation will vary depending on your locale and time zone. For example, the From Date field defaults to `12/31/69 7:00 PM` for a workstation with an en_US locale in the Eastern Standard Time (EST) time zone.
The default value for the **To Date** field is always the current date and time, formatted for your locale and time zone.

To avoid this problem, always enter your dates and times carefully, following the example provided above each field.

Using the Delete Expired Events function appears to suspend the failed event manager

If you use the Delete Expired Events button in situations where there are many failed events in the current search results, or where those events contain a large amount of business data, the failed event manager can appear to be suspended indefinitely.

In this situation, the failed event manager is not suspended: it is working through the large data set, and will refresh the results set as soon as the command completes.

Failed events are not being created

If the Recovery subsystem is not creating failed events, go through the following checklist of potential causes:

- Ensure that the wpsFEMgr application is running. If necessary, restart it.
- Ensure that the failed event manager's database has been created, and that the connection has been tested.
- Ensure that the necessary failed event destination has been created on the SCA system bus. There should be one failed event destination for each deployment target.
- Ensure that the Quality of Service (QoS) **Reliability** qualifier has been set to `Assured` for any Service Component Architecture (SCA) implementation, interface, or partner reference that participates in events you want the Recovery service to handle.

The server fails when retrieving or deleting large numbers of failed events

The server can fail if you use the failed event manager to retrieve or delete a large number of failed events at once. To prevent this problem, be sure to check the total failed event count on the main page of the failed event manager before performing search or delete actions. If you have a large number of events, do not try to retrieve or delete all of them at once. Instead, use the **Search failed events** option to return a subset of failed events that match specific criteria. You can then delete all of the events in the filtered result set without causing a server failure.

Note: There are several factors in your environment that can affect the number of failed events the server can return or delete in a single request, including other processes running on the same machine as the server and the amount of available memory.

Preventing multiple failures when a service is unavailable

If your component calls a service, and that service is unavailable, a runtime error occurs. Events associated with the error are treated, by default, as failed events. Using the store-and-forward feature, however, you can drastically reduce the number of failed events that are generated by the system.

Related concepts:

"Managing failed events in WebSphere ESB" on page 563
When an event fails in WebSphere ESB, it is stored in a database in the failed events manager. You can search for a failed event, and then view, modify, resubmit, or delete it.

Storing events

You can set up your applications so that, when a runtime error occurs during application processing, the recovery service stores the events (specifically, asynchronous requests) for later submission instead of repeatedly generating failed events. For example, if you are deploying an application that calls a Web service and that Web service is unavailable, a failed event is generated and then subsequent associated asynchronous requests are stored. Then, when the service becomes available, you can forward stored asynchronous requests for processing.

How the store-and-forward feature works:

To enable the store-and-forward feature, you set the store-and-forward qualifier on the component that will be invoked asynchronously. If that component then calls a service that is unavailable, a failed event is created and a service unavailable exception is generated. Once a failed event triggers the store function, all subsequent asynchronous requests intended for the component are stopped and stored. When the service becomes available again, you can resubmit the failed events that triggered the store and set the Store and Foward widget to forward stored events for processing.

You set the store-and-forward qualifier in Integration Designer. This qualifier can be specified on an import, a SCA export, or a component. It can be specified for all interfaces of the component, a specific interface, or a specific operation. The exception part of the qualifier can apply to any of these different levels. However, the state (store or forward), applies at the component level only.

When an application with a store-and-forward qualifier is deployed to the server, and runtime validation is enabled, a runtime validator confirms that the qualifier is specified correctly. If there are any errors during validation, the installation of the application is stopped. If there are any warnings, they are displayed in the log file and the installation of the application is allowed to continue. The rules for validation of this qualifier are specified at "Store-and-forward runtime validator" on page 588.

Component calling a Web service

Suppose you have a component that calls a Web service, as in the following illustration:

Figure 137. Flow of asynchronous request processing from an SCA module to an target service when the store-and-forward feature is used

The items in the following list correspond to the numbers in Figure 137.

1. A component calls the target Web service by way of the Web service import.

   Because this is an asynchronous call, requests are put on the queue by the calling component, where they are picked up by the import.
2. The import detects that the service is down and retries the operation.

   As shown in Figure 137, the destination queue cannot be reached and, therefore, the service cannot pick up the asynchronous requests for processing.
3. If the service is unavailable, a failed event is created. If subsequent retries fail, the import generates a service unavailable exception.

   When the service unavailable exception is generated, a failed event is created and the recovery service checks the configured policy to determine whether this exception means that the service is down.
4. If it is determined that the service is down, the asynchronous requests are placed in the store-and-forward queue.
5. Additional asynchronous requests intended for the service are stopped before they can reach the import and are stored until the service becomes available.

   Storing the asynchronous requests prevents additional failures from occurring.

   The size of the queue is configurable. Make sure you specify a size that is large enough to accommodate the asynchronous requests. If the queue is not large enough, the event store will overflow, causing failed events to be generated.

   When configuring the size of the queue, estimate how long the service might be down and consider the rate of arrival of asynchronous requests.

Important:

Once the service becomes available again, you will have failed events in the failed event manager that need to be resubmitted for processing. This does not happen automatically. Use the failed event manager to resubmit the events that triggered storing in the first place. These events will be tagged with an Activated Store event qualifier. Then, open the widget in Business Space, locate the service control point that triggered the store, and select **Forward**.

If event sequencing and store-and-forward processing are specified on the same operation or interface, the failed event that triggered the store must be resubmitted before any sequenced failed events behind it can be forwarded.

SCA module calling multiple target services

The same sequence of steps occurs when multiple target services are called from a single SCA module, but with one notable exception - if one of the target services called by the component becomes unavailable, all asynchronous requests are stopped before they reach an import and stored for future processing. The store-and-forward qualifier is specified on the component's interface.

Figure 138 shows two asynchronous requests flowing from one SCA module to multiple target services. One of the target services is unavailable, but the other is available. Subsequent asynchronous requests intended for both the first target service and the second are stopped before they reach the import and stored for future processing.

Figure 138. Flow of asynchronous request processing from one SCA module to multiple target services when the store-and-forward feature is used

Behavior of store-and-forward feature after server restart, application stoppage, or application uninstallation:

If the store-and-forward feature is activated for a component in an application and the server is restarted, or if the application is stopped or uninstalled, the store-and-forward feature will behave in one of the following ways.

- If there are events in the queue when the user uninstalls the application, the queue and all stored events are deleted.
- If the application is stopped and restarted, the store remains activated.
- If there are events in the queue and the server is restarted, the state changes from store to forward and all the events in the queue are processed.

Store-and-forward in a clustered environment:

In clustered environments, when one service control point is stopped or started, the rest of the service control points for that component (across the members of the cluster) also starts or stops.

A service control point is the place in your scenario where the store-and-forward qualifier is specified. This is the point in your scenario where storage is enabled. Each service control point has a unique name which is specified by the user.

When the store-and-forward feature is enabled in a clustered environment, things that would normally cause one service control point to start or stop causes all the service control points in the cluster to start or stop. This includes both service runtime exceptions and the manual starting and stopping of the store-and-forward feature by a user in business space.

The number of listeners configured in Business Space influences how many failed events might appear in the Failed Event Manager before storing begins. This is because it takes time for the store to activate. If multiple events fail before storing is triggered, these appear in the Failed Event Manager. All events received after the store is triggered are stored until the Store and Forward widget state is changed to **Forward** in Business Space.

The following sections describe the behavior of the store-and-forward feature in a clustered environment under various circumstances.

Store-and-forward feature triggered by an exception, activated for all servers in a cluster

When a service runtime exception is generated, the store-and-forward service checks the configured policy to determine whether this exception means that the service is down. If it is determined that the service is down, the store is started on all servers in the cluster and all new asynchronous requests are stored before they can reach their respective imports.

Suppose that you have four members in a cluster, and a service runtime exception is generated by the import on the first server in the cluster. The following illustration shows that once the store-and-forward service determines that service is down, the store is enabled for all the servers in the cluster. All additional asynchronous requests are stopped before they can reach the import and are stored until the service becomes available.

Figure 139. Behavior of the servers in a cluster when the store-and-forward feature is activated

Store-and-forward feature started in business space, activated for all servers in a cluster

When the store is started in business space for one server in a cluster, storing is activated for all the servers in the cluster.

The following illustration shows the store being started in business space for one server. After the store has started, that status is propagated to the other servers in the cluster.

Figure 140. Behavior of the servers in a cluster when the store-and-forward feature is started for one server in business space

Store-and-forward feature stopped in business space, stopped for all servers in a cluster

When the status is changed from store to forward in business space for one server in a cluster, storing is stopped for all the servers in the cluster.

The following illustration shows the store being stopped in business space for one server. After the store has stopped, that status is propagated to the other servers in the cluster.

Figure 141. Behavior of the servers in a cluster when the store is stopped in one server in business space

Store-and-forward runtime validator:

When an application with a store-and-forward qualifier is deployed to the server, a runtime validator confirms that the qualifier is specified correctly. If there are any errors during validation, the installation of the application is stopped. If there are any warnings, they are displayed in the log file and the installation of the application is allowed to continue.

The runtime validator checks the following items. Validation rules verify that the values supplied in the qualifier make sense in the context of the application, or that there are no syntactic errors.

The runtime validator logs a warning in the following situations:

- A non-SCA export has a store-and-forward qualifier
- Exceptions are not unique in their hierarchy. For example, if two neighboring interfaces have identical exception specifications, that is acceptable. However, if an interface has an exception specified, and an operation under that interface specifies an identical exception, a warning is logged.

The runtime validator logs an error in the following situations:

- If the exception name is:
  - not found in the module classpath
  - not a subclass of java.lang.RuntimeException
  - a subclass of com.ibm.websphere.sca.ServiceBusinessException
- If the value for exceptionChain is not set to one of these four valid options:
  - startFromTop
  - startFromBottom
  - OnlyTopLevelException
  - OnlyLowestLevelException
- If the configuration name for each qualifier:

  - is not defined. This value is required.
  - is not named using alphanumeric characters or an underscore. Alphanumeric characters and underscores are the only supported characters.
  - is not unique across the module
- If the validator detects that the XML document does not conform to the XML schema
- If extraneous tags are found in the XML

If you encounter a runtime validation error related to the store-and-forward function, follow the suggestions in the error message to resolve the error.

Store-and-forward feature set up:

Consider the following information when implementing store-and-forward processing.

Use IBM Integration Designer to configure a store-and-forward qualifier. The store-and-forward qualifier is used to detect when there are problems with a service and prevent multiple failed events from being sent to the failed event manager. When a service becomes unavailable, the first failed event (and possibly more) are sent to the failed event manager. However, subsequent events are stored if the system encounters a service unavailable exception that matches the one set in the store-and-forward qualifier. When the service becomes available again, stored events can be forwarded to the service for processing using the Store and Forward widget in Business Space. Failed events logged before the store was activated can be resubmitted using the failed event manager.

Note: If event sequencing is configured on the same interface or operation as store-and-forward processing, failed events received by the failed event manager (before the store was activated) must be processed before stored messages.

The most common application of the store-and-forward qualifier is to detect when a service is unavailable, but it could be used to detect other problems such as not authorized exceptions by defining different runtime exceptions for the qualifier.

You specify the store-and-forward qualifier on the interface of the component or import. It can be set at the component level or operation level and defined so as to apply to all interfaces of the component, to apply to a specific interface, or on a specific operation.

Note: The store-and-forward qualifier cannot be placed on exports, except SCA exports.

Note: You cannot define a store-and-forward qualifier for synchronous requests.

Store-and-forward asynchronous request processing can be specified on the interfaces of all import bindings.

- When setting up the store-and-forward qualifier on a component or import, keep the following tasks in mind:
- You configure the runtime exceptions.

  By default, the runtime exception is the ServiceUnavailableException.

  Note: When this exception is received, the store-and-forward functionality activates and WebSphere ESB begins storing events for future processing. If any

other exception except the specified exception is received, the store-and-forward function will not be activated and events are not stored for future processing.

For information about configuring the store-and-forward feature, see Store and forward qualifier.

Consider the following store-and-forward possibilities for a component with two interfaces and two operations on each component.

Table 56. Behavior of the store-and-forward feature based on qualifier configuration

Qualifier settings	Store-and-forward behavior
If the qualifier is set on the first operation and the failed event is generated by the invocation of the second operation	store-and-forward feature is not activated
If the qualifier is set on the first operation and the first interface and a failed event is generated by the invocation of the second operation	Only the qualifier and properties specified only the first interface take effect. The qualifier and properties specified on the first operation do not take effect.
If the qualifier is set on the second operation and a failed event is generated by the invocation of the second operation, but the headers in the event are incorrect and the information for the second operation cannot be found in the headers	store-and-forward feature is not activated and a warning is logged
If the qualifier is set on the InterfaceSet (for all interfaces)	The qualifier on the interfaceSet takes effect even if the headers in the failed event are not correct
If the qualifier is set on the first interface and the first operation and a failed event is generated by the invocation of the first operation	The qualifier and properties for both the first operation and the first interface take effect. When multiple qualifier properties take effect, then it is an OR operation between the properties.

Related information:

Error when sending event

Administering the store-and-forward feature

Use the Store and Forward widget in Business Space to change the state or configure service control point properties.

About this task

The Store and Forward widget has two states: Store and Forward. The state should bet set to Store if you want to store failed events for future processing. The first failed message is sent to the failed event manager, but subsequent failures are prevented because messages are stored until the destination becomes available again. When the service is available, stored messages can be forwarded by setting the service control point to its Forward state.

You can also see store-and-forward service control point properties. Some of these properties can be edited.

For more information related to manually starting the store or forwarding stored messages, see the help information for the Store and Forward widget: Store and Forward.

Important:
The Store and Forward widget cannot be used to configure a store-and-forward

qualifier for a component. Use IBM Integration Designer to configure store-and-forward qualifiers.

Enabling common base event generation during store-and-forward processing:

Enable the store-and-forward function to create common base events when the store is started or stopped using the administration console in WebSphere ESB.

Before you begin

You must be logged in as administrator or operator to perform this task.

About this task

Configuring WebSphere ESB to create common base events when the store is stopped or started is useful in situations where WebSphere Business Integration Monitor is used to adjust or track common base events in a monitor model. This is also true for scenarios where WebSphere Business Integration Monitor uses common base events in their key performance indicator calculation. Events that correspond to the Common Base Events specification that are generated during store-and-forward starts or stops, can be used by custom-made service monitors that are designed to track these events.

Common Base Events can be viewed in the Common Base Event browser.

To enable common base event generation during store-and-forward starts or stops, complete steps in the Procedure section.

Procedure

1. Open the administration console.
2. Select **Troubleshooting** and then **Logs and trace**.
3. Select a server and then**Change log detail levels**.
4. Select the **Runtime** tab. The **Change log detail levels** panel defaults to the Configuration tab. Making changes while the Configuration tab is selected do not take effect until server restart. Changes to the otherwise identical Runtime tab take effect immediately.
5. Expand all components and type **WBIEventMonitor.CEI.StoreAndForward.*=all**. You can also replace "CEI" with "LOG". The CEI string causes events to appear in the Common Base Event Browser, while the LOG string will cause events to appear in the trace.log file. You can select one trace string or both of them at the same time.

   Note: This string (minus the "=all" portion) might be available as part of the list of available traces. If it is, then you can select it from the list and then select **All messages and trace** instead of typing in the string.
   Sometimes the logging string does not show up in the administrative console list until the first common base event occurs. This is a known limitation in the common base event framework. If you have installed a store-and-forward enabled application, completed the steps above, and do not see the WBIEventMonitor.CEI.StoreAndForward string in the list of available traces, type the trace string (for example, WBIEventMonitor.CEI.StoreAndForward=all) in the box and tracing will be enabled even if a common base event has not been generated.

Results

Events that correspond to the Common Base Events specification are generated when store-and-forward starts queueing requests or stops queueing requests. If these settings have been made in the runtime administrative panel, changes will take effect immediately. If these settings have been made in the configuration panel, changes will take effect after you restart the server.

Chapter 6. Developing for WebSphere ESB

Developing (programming) for WebSphere ESB includes developing business integration solutions and developing modules for a service-oriented programming model.

This section also includes programming guides and examples.

Enterprise Service Bus programming

This development guide is for enterprise architects, integration architects and developers who are responsible for implementing integration and connectivity solutions. It begins with the high-level concepts and abilities that will help you to understand the capabilities and usage patterns of an Enterprise Service Bus (ESB). The development guide then describes how to implement the capabilities and patterns on WebSphere Enterprise Service Bus.

Introduction

This development guide is for enterprise architects, integration architects and developers who are responsible for implementing integration and connectivity solutions. It begins with the high-level concepts and abilities that will help you to understand the capabilities and usage patterns of an Enterprise Service Bus (ESB). The development guide then describes how to implement the capabilities and patterns on WebSphere Enterprise Service Bus.

Service-oriented architecture

Service-oriented architecture (SOA) is an architectural approach that supports the flexible integration of IT systems. It does this by providing a method to disconnect the service provider (service capability which delivers a defined action) from the service requester (client that sends the request). This enables the substitution of one service provider for another, without the requester being aware of the change and without the need to alter the architecture as shown in Figure 142.

Figure 142. Shows the connection between the service requester, service provider and ESB

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.

Services are the building blocks of SOA, providing function as a base point for building distributed systems. Services can be invoked independently by either external or internal service consumers to process single functions, or can be linked

© Copyright IBM Corp. 2007, 2011

together to form more complex functionality and so quickly devise new functionality.

The Enterprise Service Bus

An ESB is used to connect a service requester to a service provider so that messages can be routed between the two platforms. The ESB is a collection of software components that manage messaging from one part of the network to another. The ESB handles mismatches between the requesters and providers, including protocol, interface or quality of service mismatches.

The ESB processes messages exchanged between the service endpoints. A service endpoint is a location where the service is available on the network. In contrast with regular business application components, the ESB is concerned with the flow of the messages through the infrastructure and not just with the business content of the messages. Rather than performing business functions, the ESB performs mediation capabilities including routing, transformation, and logging operations on the messages.

WebSphere Enterprise Service Bus

WebSphere ESB is designed to provide an ESB for IT environments built around open standards and SOA. It delivers easy to use functionality that is built on the messaging and web services technologies of WebSphere Application Server.

WebSphere Application Server is the foundation for WebSphere ESB, providing not only the required quality of service, the J2EE runtime environment, and the messaging infrastructure, but also providing broad support for open standards and web services. WebSphere ESB is built on the Network Deployment product and provides a wide range of capabilities for large enterprise networks, including clustering, failover, and scalability features.

IBM Business Process Manager extends the capabilities provided in WebSphere ESB to allow business applications to be developed.

IBM Integration Designer, is the tool for use with WebSphere ESB. It is designed to build applications that will be deployed to either WebSphere ESB or IBM Business Process Manager .

WebSphere ESB supports connectivity between endpoints through a variety of protocols and application programming interfaces (APIs) such as:

- WebServices SOAP 1.1 and 1.2
- Java Message Service (JMS) 1.1
- WebSphere MQ
- Enterprise Java Bean (EJB) Stateless Session Bean
- HTTP
- Enterprise Information Systems (EIS) integration using WebSphere Adapters

Figure 143 on page 595 shows how WebSphere ESB is built on top of the WebSphere Application Server. Because of this, WebSphere ESB can provide smooth interoperability with other products in the WebSphere portfolio, including WebSphere MQ and WebSphere Message Broker.

Figure 143. WebSphere ESB, installed on top of WebSphere Application Server

Common Usage Patterns

There are seven common usage patterns that help to speed up the process of implementing connectivity and integration solutions. The patterns summarize some of the more common solutions in the connectivity space. These patterns are common usage scenarios that all Enterprise Service Bus (ESB) technologies should support.

"Simple service proxy pattern"
"Service selector pattern" on page 596
"Service translator pattern" on page 597
"Service gateway pattern" on page 598
"Message enrichment pattern" on page 599
"Batch aggregation pattern" on page 600
"Multiple source aggregation pattern" on page 601

Simple service proxy pattern

The simple service proxy pattern creates a one-to-one mapping between a service requester and service provider. By placing an ESB between the two services it hides the real location of the service provider. The simple service proxy pattern is also of particular value in providing a point for access control, request tracking or auditing.

You can use this pattern when:

- You want to provide access to a service through a control point without exposing the actual location (endpoint address) of the service to the clients.
- You must apply some form of control (access management, authorization, auditing or logging) every time a service is accessed.

Figure 144 on page 596 shows the simple service proxy pattern. The target service is hidden by deploying an ESB mediation which implements the same interface. This mediation acts as a virtual service (or proxy) which redirects all requests to the real service provider.

A service registry is a repository of data describing a set of services and their associated properties and behaviors. This registry can be queried by the ESB to

determine the endpoint of the service provider.

Figure 144. Simple service proxy pattern

For example, operations within a bank application often need to be audited as a legal requirement. The simple service proxy pattern can be used to implement this business requirement that ensures all requests are audited, and removes the need for the service provider to implement the capability.

Service selector pattern

The service selector pattern provides multiple implementations of the same service interface to be grouped behind a single endpoint address. Each implementation can have a different quality of service or behaviour, and each client request can be matched to a particular implementation determined by different criteria, such as the content of the message into the ESB.

You can use this pattern when:

- There are multiple implementations of a particular service all sharing the same interface.
- Different implementations provide different qualities of service.
- Requests from different clients are to be routed to a particular implementation in accordance with some set of predefined criteria.

Figure 145 on page 597 shows the service selector pattern, which implements the required logic to select a specific endpoint address based on the context or content of each client request.

A service registry is a repository of data describing a set of services and their associated properties and behaviors. This registry can be queried by the ESB to determine the endpoint of the service provider.

Figure 145. Service selector pattern

An example of when the service selector pattern would be used is if a fictitious company wants to differentiate itself from its competition by offering tiered levels of service. The company's goal is to offer delayed stock quotes to their standard customers and real-time quotes to their premium customers, who pay a fee.

The service requester application provides a query containing a stock symbol and customer ID to the content based router, which processes the query. The customer's subscription level is determined, and depending on the level of subscription, the query is routed to the appropriate service provider. The quote that is returned from the service provider is then returned to the client application. More on this scenario is found in the "Example scenarios" on page 614.

Service translator pattern

The service translator pattern provides access to a service implementation with a different interface or protocol than the interface or protocol used by the service requester. You can select operations to be restricted or restructured on some interfaces, and you can convert and format data for users of specific interfaces.

You can use this pattern when:

- You want to restrict access to certain functions on the user interface.
- The interface needs to be modified to conform to the requirements of certain departments.
- Requesting services expect to use different interfaces or protocols.

Figure 146 on page 598 shows the service translator pattern, which transforms the request and then routes it to the actual service provider. The mediation must also process the responses from the target service, transforming them back into the form expected by the client.

Figure 146. Service translator pattern

An example of the service translator pattern, is if a fictitious company, **Personal Holidays** book flights, and the flight availability must be checked against a fictitious airline company, **My Airways**, flight schedule. The Personal Holidays booking system represents information differently to the My Airways booking system. The service translator pattern within the ESB mediation is able to transform incoming messages from the service requester into the format that My Airways represent their information. More information about this scenario is found in the "Example scenarios" on page 620.

Service gateway pattern

The service gateway pattern introduces a point of control and can bring a range of services under the control of an ESB without the need to develop individual mediations for each service. The service gateway provides a single access point and acts as a proxy for multiple services. In addition, service gateways encapsulate transformations, routing, and common processing across all the services.

You can use this pattern when:

- You want to route messages to multiple endpoints using a variety of protocols.
- A set of services needs to be accessed in the same way without the need for multiple mediations.
- You want to provide generic processing (for example auditing, security enforcement point) across all the services.

Figure 147 on page 599 shows the service gateway pattern. Messages enter the gateway and the target service is identified. Common processing is completed (such as auditing) before the message is routed onwards to an end provider.

A service registry is a repository of data describing a set of services and their associated properties and behaviors. This registry can be queried by the ESB to determine the endpoint of the service provider.

Figure 147. Service gateway pattern

For example a bank have a number of completely different services that need to be audited as a legal requirement. They want the same level of audit across all the services. The service gateway pattern is used to implement this business requirement.

Message enrichment pattern

The message enrichment pattern provides a method to modify or add to the existing content in a message before it is forwarded onto the service provider. An example of message enrichment is to retrieve data from a database and add the data to a given location in a message.

You can use this pattern when messages are sent, in order to obtain further information from a database.

Figure 148 on page 600 shows the message enrichment pattern. The message that arrives into the ESB does not contain all the data required by the service provider. Data is added by the ESB before the message is sent to the service provider. For example, the service requester only knows the customer ID and product ID. Message enrichment provides the full product and customer details for the order to be processed.

Figure 148. Message enrichment pattern

An example of when the service enrichment pattern can be used is if a particular field does not exist within a message structure. A Database Lookup mediation primitive can be used to establish the details of the field in any given instance. More information about this scenario is found in the "Service enrichment" on page 622.

Batch aggregation pattern

The batch aggregation pattern provides a method for an inbound request to be mapped into several individual outbound requests to the service provider. The responses from these requests can be collected (aggregated) into a single response for the original request.

You can use this pattern when there is a need for messages that include multiple records and each of which can be forwarded onto the service provider individually.

Figure 149 on page 601 shows the batch aggregation pattern, where the inbound message to the mediation is split into several outbound requests to the service provider, and then collected back into a single message to send back to the service requester. The service requester receives all the relevant information required in one batch message.

Figure 149. Batch aggregation pattern

Multiple source aggregation pattern

The multiple source aggregation pattern provides a method for an inbound request to be mapped into several individual outbound requests to multiple service providers. The responses from these requests can be collected (aggregated) into a single response for the original request.

You can use this pattern when a single message is sent into the ESB that requires two or more service providers to process the message, or parts of the message. An aggregated response is sent back to the service requestor.

Figure 150 shows the multiple source aggregation pattern, where the inbound message to the mediation is split into several outbound requests for several service providers. The results are then collected back into a single message to send back to the service requester. You can use this pattern if the inbound request needs information from many different service providers, for example a quote comparison web site.

Figure 150. Multiple source aggregation pattern

Service Component Architecture

Service Component Architecture (SCA) is a model to define and implement SOA based solutions. This section details the IBM implementation of SOA within WebSphere ESB, and its associated tooling, Integration Designer.

There are two defined sections within SCA:

1. Service assembly, which details the service definitions and relationships of the export, import, components and interfaces

2. Service implementation, which details the construction of the service definition and the transport of information through defined protocols (web services, JMS, MQ, adapters).

In an SOA framework, high-level business components are exposed as services. SCA provides a model for how the services in an SOA framework are structured.

Module

A module is a unit of deployment, where services are packaged together. Components within a module are assembled for performance, and can pass their data by reference. A module can be seen as a scoping mechanism; it sets an organizational boundary for services.

Modules can include:

- Service components, that hold the business function within a module. A service component configures a service implementation. The service component name inside a module is unique. A service component can have an arbitrary display name, that is more useful to a user. It is not necessary for each module to have a service component, but there is no limit to the number of service components per module.
- Imports, that are calls to services external to the module. It is not necessary for each module to have an import, but there is no limit to the number of imports per module. Imports are detailed later on in the chapter.
- Exports, that are used to expose components to callers that are external to the module. It is not necessary for each module to have an export, but there is no limit to the number of exports per module. Exports are detailed later on in the chapter.
- Interfaces, which can be associated with one or more components. The interfaces associated with a service component show the business operations associated with the component.
- Stand-alone references or reference applications that are not defined as SCA components (for example, JavaServer Pages). These applications can then interact with SCA components. There can be only one stand-alone reference per module.
- Other elements, for example WSDL files, Java classes, XSD files.

Figure 151. Structure of a module

Figure 151 shows an example module, which contains two service components, each containing an implementation. An export reveals Service Component A's capabilities to external service consumers. A stand-alone reference shows the component's capabilities to other non-SCA artifacts included within the deployment artifact such as JavaServer Pages (JSP). Service Component A may require the capabilities of two other services. One implemented by Service Component B, and one included using an SCA import. All exports, imports and components have interfaces that describe the inputs and outputs of these elements. References are used on service components and stand-alone references to declare the additional capabilities they require to function. Wiring is used between all the elements to represent their relationships.

SCA implementation in Integration Designer

You can build SCA modules within Integration Designer. Modules contain all the artifacts required to create a complete SCA application and they can be organized into projects. Additionally SCA libraries can be created in Integration Designer. You can build libraries to store artifacts, and you can share them between multiple SCA modules. Libraries are detailed later on in the chapter.

There are two types of modules that are supported by Integration Designer:

1. A business module that contains a choice of many component types, often used to support a business process.
2. A mediation module, which contains one or more mediation flow components. It is not necessary for a mediation module to contain any Java components, but there is no limit to the number of Java components that it can contain. Java

components augment the mediation flow component. The mediation module acts as a gateway to existing external services, which is common in ESB architectures.

A business module is deployed to an IBM BPM server. A mediation module is deployed to either an IBM BPM server or a WebSphere ESB server. Both types of modules support imports and exports.

Mediation Modules

Mediation modules are SCA modules that can change the format, content, or target of service requests. Mediation modules operate on messages that are in-flight between service requesters and service providers. Mediation modules can route messages to different service providers and can also transform the message content or structure. Mediation modules can provide functions such as message logging, and error processing that are tailored to your requirements. You can change certain properties of mediation modules, from the WebSphere ESB administrative console, without having to redeploy the module. Figure 152 shows a simplified example of a mediation module. The mediation module contains one mediation flow component.

Figure 152. Mediation module

Module imports and exports

The mechanisms used for invocations from module to module and module to external services, are called imports and exports. Imports and exports are represented from the point of view of the module. The module is a self-contained bundle of components that perform a specific business function. A module invokes this capability using an import if it is required to use the business function from another service to call a function (external service or other module).

Exports provide the ability to make a service available over a number of different transport protocols. The export is associated to a particular component within the module.

Imports and exports also have the ability to interact with service providers across a variety of protocols and Application Programming Interfaces (APIs), such as:

- Web services

- HTTP
- JMS
- MQ
- EJB
- WebSphere Adapters
- SCA

Data handlers

Data handlers are reusable transformation logic in WebSphere ESB, which can be used by exports and imports. Data handlers transform native formats into business data and from business data into native formats required by external services.

In your SOA implementation, business data can flow between service providers and service requesters over a variety of protocols (HTTP, JMS, MQ, and so on), in a variety of data formats such as comma separated value, delimited, fixed width, COBOL and so on. Different protocols can have the same mechanisms for carrying the business data in their respective protocol envelope. For example, a HTTP message can encode its data using a comma delimited format, which can also be used in a JMS message. The data handler interface is protocol neutral, which enables data handlers to be usable across all of the protocols supported.

WebSphere ESB supplies a number of built in data handlers including:

- Atom feed format
- Delimited format
- Fixed width format
- JavaScript Object Notation (JSON) format
- SOAP data handler
- WTX Invoker Data Handler
- WTX MapSelection Data Handler
- XML Data Handler

Function selector

The function selector determines which operation defined on the associated interface is invoked called. When a message arrives at an export it will include raw transport header information and encoded data for the body of the message. The function selector can use header data, body data, or both, to determine which of the operations on the associated interface can be invoked. As this mechanism is likely to be dependent on the protocol headers, the implementation of the function selector will be protocol specific. For example, a JMS message header can specify the operation name and a function selector can use this JMS message header to determine the operation that is called.

SCA components

SCA components are building blocks for mediation modules. There are two types of SCA components, which are allowed within a mediation module:

1. Mediation flow component
2. Java component

Mediation flow component

Mediation modules contain a specific type of SCA component called a mediation flow component. A mediation flow component manipulates the message as it flows through the mediation module. A mediation flow component can be associated with several interfaces. A mediation flow is the process that occurs for a particular operation on the interface. There is at least one mediation flow for each operation on the interface.

An interface can have two types of operation:

1. One way (request only), with a single request flow that holds the logic for the request message.
2. Two way (request/response) with a request flow and a response flow. The request flow holds the logic for the request message. The response flow holds the logic for the response message.

A mediation flow can contain any number of mediation primitives. Mediation primitives are reusable blocks of function that are wired together to build a mediation flow. These mediation primitives can route and transform the message. WebSphere ESB supplies a set of mediation primitives that provide functionality for message processing. You can also contribute your own primitives. If the mediation module will only transform from one protocol to another, a mediation flow component is not required. For example, a service request might be received using SOAP/JMS but might need transforming to SOAP/HTTP before sending on. In this instance, a Web Service Export (SOAP/JMS) and Web Service Import (SOAP/HTTP) is required.

Java component

The Java component provides an alternative implementation option for creating SCA components within WebSphere ESB. You can use Java code, instead of the mediation flow component to manipulate the messages as they pass through the system. This can provide additional flexibility in the solutions you implement.

Library

A library is a project that is used to store resources and artifacts, so that the resources or artifacts can be shared by multiple modules defined in the library. The usage of libraries is good practice where interfaces are shared across multiple mediation modules.

Unlike modules or mediation modules, library projects can share resources. The resources or artifacts that are defined in the library are:

- Business objects
- Interfaces
- Mediation subflow
- Datamap
- Relationship
- Binding resource

There are two kinds of sharing:

1. Sharing of artifacts for development
2. Sharing of artifacts within the solution at run time

For a module or mediation module to use the resources from a library at run time, you must add a dependency, indicating that the module is dependent on the

library. You can add a library to the module and select to deploy it with the module. You can also add library dependencies to a library; for example, if one library uses resources in another library, then you must add the library dependency.

Artifacts in a library are public, and they can be shared across modules. Artifacts in a module are private. WebSphere ESB integration developers should organize artifacts by taking into consideration logical function and visibility. For example, related data types that are used by various pieces of a system should be placed into a library. Other modules can then easily reuse this library. Only one copy of each element needs to be created and maintained using this method. Likewise, related business logic features of the application should be grouped into modules. Structuring an application in this way will yield a more flexible, easily maintained application. It will also be easier to add new features to the application.

Business objects and interfaces

Business objects are containers for application data, such as customer details or an invoice. Data is exchanged between components by way of business objects. When using imports and exports in assembly diagrams, it is good practice to put the business objects and interfaces that are used by the import and exports into a library so that they can be shared. Then, add a dependency on the library to all of the modules that use these common resources. Avoid copying the same business objects and interfaces into different modules to use them.

Mediation subflows

If you have mediation logic that you want to reuse across mediation flows you can define a mediation subflow within a library. Mediation flows cannot be placed in a library.

Library resources that are shared

If you choose to deploy a library with a module, which is the default setting, a copy of the library is made for each module when the module is deployed. After deployment, if shared resources change in the library, all modules using the resources have to be updated. For example, two modules share some resources in a library. The applications are deployed. One of the modules has to be updated, resulting in changes to some of the shared resources in the library. In this case, the second module also needs to be updated to reflect the changes in the shared resources.

Mediation Flow Component

Mediation flows operate on messages that are in-flight between the service requester and service provider. Mediation primitives are the building blocks of mediation flows. You create mediation flows in mediation flow components. Mediation flow components can exist in either modules or mediation modules.

Mediation flows

A mediation exposes an interface to the service requester. The mediation might use another interface to call the service provider. The interface used by the service requester is referred to as the source interface, and the interface provided by the

service provider is referred to as the target interface. Depending on the service requester and the service provider, the source and target interfaces might be different.

For each source operation, the mediation can contain a request flow. For each operation that has a response, the mediation has a response flow.

A request flow processes a request message from a service requester. The flow begins with a single input node for the source operation, followed by one or more mediation primitives wired together. Each target operation has a callout node, and the request flow can be wired to the callout node to call a particular operation. If a message is to be returned to the source directly after processing, the request flow can be wired to an input response node in the request flow. If fault messages are defined in the source operation, an input fault node is also created. Wiring the request flow to an input fault node allows a fault message to be returned directly to the service requester.

A response flow processes responses returned from the service provider. The flow begins with a callout response node for each target operation, followed by one or more mediation primitives wired together. The response flow contains a single input response node representing the source operation. Wiring the response flow to the input response node causes a response to be sent to the service requester. If fault messages are defined in the target operation, a callout fault node is also created in the response flow. The callout fault node allows fault messages that are returned by the target operation, to be processed. Errors that are returned by the operation but are not defined as fault messages, are propagated to the fail terminal of the callout response node.

A mediation flow component has an error flow for each source operation. The error flow acts as a catch-all for messages that are propagated from any unwired fail terminal on any mediation primitive or node, in either a request or a response flow.

Figure 153 shows the three two flows within a mediation flow component.

Figure 153. Mediation flows

Mediation primitives

Each mediation primitive allows you to manipulate messages. For example, you can route a message or change its content. Mediation primitives process messages as Service Message Objects (SMOs).

You can use mediation primitives to change the format, content, or target of service request messages.

All mediation primitives have an input terminal (called in) that can be wired to accept a message. Most mediation primitives have one fail terminal (called fail) and one or more output terminals. However, the Stop and Fail mediation primitives have no fail terminal and no output terminals.

If a failure occurs during the processing of the input message, the fail terminal propagates the original message, together with any failure information. If the fail terminal is not wired, the error flow runs. If the error flow has not been implemented, the running of the flow stops and a failure is reported to the system log.

If an output terminal of a mediation primitive is left unwired at run time, the unwired output terminal stops this path in the flow without generating a failure, and the message is consumed.

The behavior of a mediation primitive is controlled by properties. Each mediation primitive type has its own properties that configure the mediation primitives' capability.

Administrative control of mediations

Mediation primitives and nodes have properties that can be used to customize their behavior. Some of these properties can be made visible to the WebSphere ESB administrator by promoting them. Certain properties lend themselves to being administratively configured. Integration Designer lists the properties that you can choose to promote under the promoted properties of a mediation primitive.

Promoted properties are given an alias name, and you can set the alias name so that it is meaningful in the context of a particular mediation. The alias name is the property name that is displayed on the WebSphere ESB administrative console; multiple promoted properties can be given the same alias name if they are of the same type. Therefore, what appears as a single property, in the administrative console, can set the same value in multiple mediation primitives. You can set the value of a promoted property, from Integration Designer, during installation of the module and once the application has been installed using the WebSphere ESB administrative console.

Service Message Objects

The Service Message Object (SMO) provides an abstraction layer for processing and manipulating messages exchanged between services.

Data passing through mediation flows is represented by the SMO. The SMO is considered to be an envelope containing the following groups of data:

- Context information (data other than the message payload and headers).
- Header information associated with the message. For example, Java Message Service (JMS) headers if a message has been conveyed using the JMS API, or MQ headers if the messages has come from WebSphere MQ.

- The body of the message: the message payload, is the application data exchanged between service endpoints.
- Message attachments

The SMO is a DataObject and its content can be accessed via the Service Data Object API (SDO API). SDO is a standard for representing structured business data. The body of the SMO contains a business object representing the application payload.

SMO structure:

All SMOs have the same basic structure. The structure consists of a data object called a ServiceMessageObject, which contains other data objects representing the header, body, attachments, and context data.

The structure of a SMO is shown in Figure 154 on page 611.

Figure 154. Overview of SMO structure. The context, headers, body and attachments of a ServiceMessageObject

The ServiceMessageObject Java interface provides access to the message headers, message body, message context and message attachments.

SMO headers

The SMO headers carry header information for different types of messages.

The SMO headers contain information that originates from a specific import or export binding (a binding specifies the message format and protocol details). Messages can come from a number of sources, so the SMO has to be able carry different kinds of message header. The kinds of message headers include:

- SOAP
- HTTP
- JMS
- WebSphere MQ
- SCA
- Web services
- WebSphere Adapters

SMO context

The SMO context lets mediation primitives pass data that is not part of the message payload, between themselves.

SMO context objects are either user-defined or system-defined.

System-defined context objects

The following context objects are system-defined:

- failInfo. The failInfo context object is used to hold exception information
- primitiveContext. The primitiveContext object contains information used by specific mediation primitives
- dynamicProperty. The dynamicProperty object is used to override promoted properties

User-defined context objects

User-defined context objects can be used to store properties that mediation primitives use later in the flow.

The following context objects are user-defined:

- correlation context is used for mediation primitives to pass values from the request flow to the response flow.
- transient context is used for passing values between mediation primitives in the current flow. Values in the transient context are not shared between the request and the response flows or between branches of the same flow.
- shared context is a storage area you can use if you want to aggregate data. The shared context is shared between all branches of a flow. It is not shared between the request and the response flows or between branches of the same flow.
- user context is slightly different to other user-defined context objects. It is used to pass data that is not part of the message payload, between multiple SCA components in the same module or across SCA module boundaries.

SMO body

The body of a SMO is a business object that is defined by reference to a Web Services Description Language (WSDL) message.

For each part defined by the WSDL message there is one element under the SMO body. The contents of the SMO elements are of the structure of the WSDL part definition. The element names depend upon the kind of WSDL message that the SMO is defined from.

SMO attachments

The SMO contains an `attachments` element for each attachment associated with a SOAP message.

The SMO `attachments` element let you send and receive SOAP messages that have attachments of various types.

You might want to send SOAP messages with attachments and let the attachments pass through the mediation flow unchanged, or you might want to create new attachments, perhaps from information in the message or from an external source.

Data representation of SMO:

A SMO has an SDO and XML nature and behaves like a business object. This means that you can always obtain XML representations of the SMO. A SMO can also be treated as a business object.

You can use the SMO with business object utilities such as BOCopy and BOXMLSerializer. In addition if you need to examine the SMO, you can access and manipulate it using the BO API (SDO API). This provides extensive capabilities including the ability to get and set individual fields. The SMO also has its own Java interface which is a convenience method for accessing the top level structure of the SMO.

Many mediation primitives have properties that contain XPath expressions. You can use this XPath expression to specify which subset of the message for the mediation primitive, should be processed. Depending on the mediation primitive, you can specify: `/`, `/body`, `/headers`, `/context`, `/attachment`, or your own XPath expression. `/` refers to the complete SMO, `/body` refers to the body section of the SMO, `/headers` refers to the headers of the SMO. If you specify your own XPath expression, the part of the SMO you specify is processed.

Integration Designer shows the structure of a message and allows you to select locations within the message. In this way you can navigate the structure of a message and create XPath expressions.

Business objects provide a platform wide data representation. The structure of a business object is defined in an XML schema which is either imported or created in Integration Designer using the business object editor. Business objects are used to represent the data that is passed through an SCA module.

Routing Messages Within a Mediation Flow

Within WebSphere ESB, there are several mediation primitives that can be used for routing messages in a mediation flow.

Most mediation primitives have one input terminal (in), one fail terminal (fail), and one or more output (out) terminals, as shown in Figure 155. The in terminal is wired to accept a message and the other terminals are wired to distribute messages. If a failure occurs, the fail terminal distributes the original message, together with any failure information.

Figure 155. Mediation primitive

Figure 155 shows that a mediation primitive with two out terminals and one fail terminal. A mediation primitive can be wired within mediation flows to specify the potential paths to which a message can be distributed. The actual path for a message is dependent on the mediation primitive. There are different paths a message can take within a mediation primitive:

- Messages might leave an out terminal one or more times.
- Messages might leave the fail terminal once.
- Messages might not leave any of the terminals.

Flow branching

Figure 156. Branched mediation primitive

Figure 156 shows how a mediation primitive can have multiple wires from one output terminal branching to several services. You cannot specify the order in which messages will leave the terminal. If you need to send the message through a specific wire first, then you must use multiple output terminals each with a single wire. You can also select a mediation primitive that will allow you to control the order in which output terminals are invoked (for example see Figure 159 on page 617).

Example scenarios

This section describes example scenarios that a fictitious company want to implement. It then goes on to discuss the different mediation primitive options that can be used in the implementation.

The scenario addresses the business need of a financial services company that provides an interactive web-based stock market service to its customers.

The company wants to differentiate itself from its competition by offering tiered levels of service. The company's goal is to offer delayed stock quotes to their standard customers and real-time quotes to their gold customers, that is, customers who pay a fee.

The service requester provides a query containing a stock symbol and customer ID to the mediation primitive, which processes the query. The customer's subscription level is determined, and depending on the level of subscription, the query is routed to the appropriate service provider. The quote that is returned from the service provider is then returned to the client application. The following areas look at the ways in which the query can be processed by different mediation primitives.

Message filter mediation primitive

To facilitate the basic request, the company can use the Message Filter mediation primitive within their mediation flow component to selectively route messages that meet a certain criteria, through different paths of a flow, using XPath expressions.

Figure 157. Message filter mediation primitive

Figure 157 shows how the gold output, is associated with the real-time service provider, to offer real-time stock quotes to all gold customers. The default output wire is associated with the delayed service provider, to give delayed stock quotes to all standard customers. The contents of the input message are compared with each XPath expression in turn, and depending on which condition is met, a message can be sent from the associated output terminal.

```
<body>
 <getStockQuote>
  <companyID>IBM</companyID>
  <customerType>GOLD</customerType>
  <person xsi:type="Customer">
   <customerID>013652</customerID>
   <customerName>Joe Bloggs</customerName>
   <customerAddress>123 One Street</customerAddress>
  </person>
 </getStockQuote>
</body>
```

The previous input message is received by the Message Filter mediation primitive. The following XPath expression is used to signal that the message is sent from the gold output wire:

```
/body/getStockQuote/customerType=='GOLD'
```

If this condition is not met, a message is sent from the default terminal.

Type Filter mediation primitive

The company want to extend the functionality of their mediation flow component, by giving employees the same real-time quotes that the gold customers receive.

To facilitate this request, the company use the Type Filter mediation primitive within their mediation flow component. The primitive uses XPath expressions that can route messages down different paths of a flow, based on their business object type. The Type Filter mediation primitive always sends a message from the first matching output terminal.

Figure 158. Type Filter mediation primitive

Figure 158 shows that the message that is sent from the Message Filter default output wire to the Type Filter mediation primitive. If the business object of type employee is found; the message is sent along the employee output wire to the real-time service provider. For any other business object the message is sent along the default output wire to the delayed service provider.

```
<body>
 <getStockQuote>
  <companyID>IBM</companyID>
  <customerType>STANDARD</customerType>
  <person xsi:type="Employee">
   <employeeID>013690</employeeID>
   <employeeName>Jack Bloggs</employeeName>
   <employeeAddress>123 Two Street</employeeAddress>
  </person>
 </getStockQuote>
</body>
```

The previous input message is received by the Type Filter mediation primitive. The following XPath expression and business object type are used to signal that the message is sent from the employee output wire:

```
XPath: /body/getStockQuote/person
Business object type: Employee
```

Flow Order mediation primitive

The company must log all requests that go to the real-time stock service provider.

To facilitate this request the company use the Flow Order mediation primitive. With this primitive, the company can order the output of the messages.

Figure 159. Flow Order mediation primitive

The log must be carried out before the messages are sent to the real-time service provider through the callout. The log service is a one-way operation. Figure 159 shows that the message that is output from both the Message Filter and the Type Filter mediation primitives is sent to the Flow Order mediation primitive. The output that is sent includes information of all gold member customers and employees that have requested real-time stock quotes. This mediation primitive will send a message to the Service Invoke Log, before sending a message to the callout.

Custom mediation primitive

The company have introduced a silver level of subscription. Silver members will receive real-time stock quotes between the hours of 9:00am and 12:00pm. Outside these hours, silver members receive delayed stock quotes.

Figure 160. Custom mediation primitive

The enhanced logic to provide time dependent selection of the service is not possible with the Message Filter or Type Filter mediation primitives. In this case, a Custom mediation primitive is used to provide more flexibility. Using a Custom mediation, the entire message can be accessed using a Java API. Logical expressions can be created using Java. The following Java code shows how the Custom mediation is created:

```
 String customerType = smo.getString("/body/getStockQuote/customerType");
if(customerType.equals("GOLD") || (customerType.equals("SILVER") &&
Calendar.getInstance().get(Calendar.HOUR_OF_DAY)<12 &&
Calendar.getInstance().get(Calendar.HOUR_OF_DAY)>8))
{
 // Fire realtime service output service
```

```
 realTimeTerminal.fire(smo);
}
else
{
 delayedTerminal.fire(smo);
}
```

Data Transformation

Within a mediation module, the message is received from a service requester before it is sent to a service provider. The structure of the message from a service requester is not always the same as that of a service provider. One of the common uses of a mediation is to transform data from one format to another.

Business Object Map and XSL Transformation mediation primitives

There can be structural differences in the types of data within the source and target XML message body type.

An example is shown in Figure 161 on page 619. The only way that the two applications can communicate, is by using a transformation. Data transformation takes place within the mediation flow component. You can transform the entire message or a particular subsection of a message. When transforming the structure of a message, the two most useful mediation primitives are:

1. XSL Transformation (XSLT) mediation primitive
2. Business Object Mapper (BO Mapper) mediation primitive

Figure 161. Hover pane showing the source and target XML message type

Both the XSLT and the BO Mapper mediation primitives transform the SMO. The difference is that the XSLT mediation primitive changes message body content according to an XSL style sheet. The BO Mapper mediation primitive changes the

message body content, according to a BO Map which is created in the Integration Designer. Both of these mediation primitives allow services that use different business objects to communicate and exchange data.

Within the Integration Designer tooling, the Mapping Editor makes populating a target XML document with data from a source XML document a visual task. The Mapping Editor is used by both the XSLT and BO Mapper mediation primitives. The source message structure for the interface and the target message structure for the interface are within the Mapping Editor. By drawing connections between the source and target fields, you create mappings between the source and target elements and attributes.

Within a transformation of the entire SMO, you can store data outside of the message body, for retrieval by downstream primitives. For example, the context of the SMO can be used to store or retrieve data from either the transient context if it is required in the same flow (request or response) or the correlation context if it is needed in both request and response flows.

Comparison between the XSLT and BO Mapper mediation primitives:

There is some overlap between the capabilities of the BO Mapper and the XSLT mediation primitives. The following table summarizes the different capabilities of each.

Use XSLT when	Use BO Maps when
You have an existing stylesheet or want to use custom XSLT in the map.	You have an existing BO Map or submap.
You want to use XPath expressions inside the map.	You want to use the relationship service (particularly dynamic relationships).
You want to use Java snippets that use the DOM API to access or update data.	You want to use Java snippets that use the BO API (SDO API) to access or update data.
You want to use built-in XPath or EXSLT functions to access or modify data.	You want to order the completion sequence for the transforms.
You want to combine contents from more than one array (repeating element).	BO Maps can be faster when used with WebSphere Adapters, because the BO Maps work directly with the SDO API.

When performance is important and the advanced functions offered by XSLT are not required, the BOMapper mediation primitive should always be used.

Example scenarios:

In this scenario, a fictitious company, **Personal Holidays** have a holiday booking system.

In order for customers to book flights with Personal Holidays, the flight availability must be checked against a fictitious airline company, **My Airways**, flight schedule. The interface for the Personal Holidays booking system is generic and not airline specific. The Personal Holidays booking system represents information differently to the My Airways booking system. The differences are shown in the following XML.

The Personal Holidays flight booking system message that represents a flight search:

```
<flightBooking>
 <departFrom>London</departFrom>
 <goingTo>Paris</goingTo>
 <departDate>29/10/2010</departDate>
 <returnDate>12/11/2010</returnDate>
 <adults>2</adults>
 <children>0</children>
</flightBooking>
```

The My Airways flight booking system message that represents a flight search:

```
<flightBooking>
 <countryOfDeparture>UK</countryOfDeparture>
 <from>London</from>
 <to>Paris</to>
 <depart>29/10/2010</depart>
 <return>12/11/2010</return>
 <flightClass>2</flightClass>
 <adult>2</adult>
 <children>0</children>
</flightBooking>
```

Mapping message structures

To implement the basic request, Personal Holidays are required to map their message structure with the My Airways message structure. To map these interfaces, they must use a data transformation. The transformation takes place on the body of the message within a simple mediation flow component. The integration developer uses an XSLT mediation primitive to transform the body of the message. This is shown in Figure 162.

Figure 162. Message Structure mapping using an XSLT mediation primitive within a mediation flow component

When an XSLT mediation primitive is created, a mapping file associated with the primitive is created or selected that describes the transformation between the source and the target business objects. This is shown in Figure 163 on page 622. The image also shows the way the values in the source message are used to populate the target message.

Figure 163. Message Structure mapping within Integration Designer

Service enrichment

Personal Holidays must determine the country of departure dynamically from within the mediation module.

The **countryOfDeparture** field does not exist within the Personal Holidays flight booking message structure. A Database Lookup mediation primitive is used to establish the country of departure in any given instance. The Database Lookup mediation primitive populates the country of departure into the transient context. In this example, the company use a BO Mapper mediation primitive to transform the transient context and body into the new required message structure. This is shown in Figure 164.

Figure 164. Service enrichment using a BO mapper within a mediation flow component

Data field level

In addition to changing the structure of a message, data transformation often requires changing the content of individual values. For example, it is common for date and time formats to differ between applications, or for string values to carry composed values that need to be extracted and separated. Within the mapping transformation primitives, you can operate on data values using a variety of built-in functions or calling out to custom code in a number of forms.

- The XML mapping editor provides built-in string, date/time, math and boolean functions, and allows you to call out to Java code that can use the DOM API in order to compose XPath expressions, or call out to other XSL templates. XML maps also provide the ability to look up values from external files or other sources, based upon a key taken from the message being transformed.

- The BO Map editor provides string extraction and link functions, and allows you to call Java that can use the BO (SDO) API. From a BO Map, you can also use the relationship transforms to manage sets of correlated data.

As an example, the Personal Holidays booking request includes a departure date of the form, yyyy-mm-dd (2010-11-10). My Airways expects the same date value to be represented in the form, dd-mm-yyyy (10-11-2010). In an XML map, the Format date function can apply a chosen pattern to a date field. This is shown in Figure 165 and Figure 166.

Figure 165. Format date function

In the Properties view, you should select the intended output format:

Figure 166. The configuration of the format date transformation

The value that is input to the function is the source field for the mapping. The transformed value is set into the output field. The transformation is performed in the XML domain, (inside the XSLT processor). The element value appears in XML. By contrast, operations that take place inside the BO Maps are performed in the SDO domain, that is using the Java business object representation. To perform an equivalent transformation in a BO Map, we would use a custom Java snippet, which would be passed to the data field as a **java.util.Date** object. This is shown in Figure 167 and Figure 168.

Figure 167. Custom transformation in a BO map

The Java code that implements the custom mapping is written in the Properties view:

Figure 168. Java code in the Properties view

In Figure 168 on page 623 the **java.text.DateFormat** class is used to format the incoming **Date** object to produce a formatted string using the built in **SimpleDateFormat.SHORT** format. This corresponds to dd-mm-yyyy or yyyy-mm-dd. By using a Java snippet, you can access the source and target data fields as local variables. These are described in the generated comments produced by the editor. From a custom Java snippet, it is also possible to invoke code from other Java classes that are contained in the module or in an associated library project.

Custom mediation

There are cases where it is difficult to represent the necessary data transformation using a BO Mapper or XSLT mediation primitive. In these circumstances, you can use a Custom mediation.

This example describes an instance of when this could occur.

My Airways have created a new service that is used to calculate the number of regular meals and vegetarian meals required for a flight.

Customers are asked to select meals as a step in the flight booking process. This information will be requested for processing as a batch process to determine the number of regular and vegetarian meals required for a flight.

The following XML shows how the business object **FlightInfo** can contain a number of **SeatBooking** records. Each **SeatBooking** has a details field which is either a **SeatBookingDetails** object (normal customers) or a **FrequentFlyerDetails** object (frequent flyer customers). Each contains the customer's meal preference, but in different named fields, **mealType** and **mealPreference**. In this case, a Custom mediation is used to provide more flexibility to transform this message body.

```
...
<FlightInfo>
 <SeatBooking>
  <type>Economy</type>
  <details xsi:type="ns0:SeatBookingDetails">
   <mealType>regular</mealType>
   <otherDetails>...</otherDetails>
  </details>
 </SeatBooking>
 <SeatBooking>
  <type>Economy</type>
  <details xsi:type="ns0:FrequentFlyerDetails">
   <customerNumber>44906</customerNumber
   <seatPreference>Aisle</seatPreference>
   <mealPreference>regular</mealPreference>
   <milesTotal>490020</milesTotal>
   <dateJoined>2009-11-09-09+00:00</dateJoined>
   <lastUpdate>2010-02-11+00:00</lastUpdate>
  </details>
 </SeatBooking>
 <SeatBooking>
  <type>Business</type>
  <details xsi:type="ns0:SeatBookingDetails">
   <mealType>vegetarian</mealType>
   <otherDetails>...</otherDetails>
  </details>
```

```
 </SeatBooking>
 ...
</FlightInfo>
...
```

The Custom mediation code completes three steps:

1. It produces a new output SMO
2. It calls another Java class to calculate the total meals
3. It sets the total meals into the SMO body

Figure 169 shows the Custom mediation in Integration Designer. The Custom mediation needs to create an output SMO that has a body type **getMealTotalsResponseMsg** in the target namespace **http://Airline/GalleyFulfillment**.

The message type and structure can be inspected using the hover pane within Integration Designer. The hover pane appears when the mouse is moved over the primitive and the on the node is clicked once.

Figure 169. Hover pane showing the message type and structure

The following code shows how to create a new SMO body of a given message type. The BOFactory **createByMessage** is used:

```
BOFactory boFactory = (BOFactory) ServiceManager.INSTANCE.
 locateService ("com/ibm/websphere/bo/BOFactory");
DataObject newBody = boFactory.
 createByMessage ("http://Airline/GalleyFulfilment", "getMealTotalsResponseMsg");
```

The following Java code allows you to set a new body for the SMO.

```
smo.setBody (newBody);
out.fire (smo);
```

Though this code has created a SMO with the correctly typed body object, it has not been populated. The SMO body is a DataObject whose immediate child elements are defined in the set of parts defined in the WSDL message for the operation.

Figure 169 on page 625 shows the body structure of the SMO. The expanded view shows that the SMO body has a child element called **getMealTotalsResponse**. This has a child element **output1** of type **MealTotals**.

To complete the code for the Custom mediation, we call the utility method **getMealTotals** (which is provided by an external Java class, enabling re-use of the utility method **getMealTotals**, perhaps in a Library) and set the processed DataObject into the **output1** child property of the **getMealTotalsResponse** DataObject. The DataObject doesn't exist in the body, it is created before setting the final result into it as shown in the following code:

```
BOFactory boFactory = (BOFactory) ServiceManager.
 INSTANCE.locateService ("com/ibm/websphere/bo/BOFactory");
DataObject newBody = boFactory.
 createByMessage ("http://Airline/GalleyFulfilment", "getMealTotalsResponseMsg");

DataObject mealTotals = MealCalculator.getMealTotals (smo.getDataObject
 ("/body/getMealTotals/input1"));
DataObject totalsResponse = newBody.createDataObject ("getMealTotalsResponse");

totalsResponse.setDataObject ("output1", mealTotals);

smo.setBody(newBody);
out.fire (smo);
```

Figure 170 shows how the correct path inside the input of the SMO body can be found using the hover pane in Integration Designer.

Figure 170. Hoverpane showing the body structure of the SMO

Note: As an alternative approach, Custom Java snippets can be used within XSLT and BO maps to complete functionality.

Message Element Setter and Data Handler mediation primitives

In addition to the BO Mapper, XSLT and Custom mediation primitives, there are other data transformation mediation primitives available within WebSphere ESB. These transformation primitives do not change the entire message structure of the SMO, they transform the content of a particular subsection of the message.

When transforming a subsection of a message, two mediation primitives that can be used are:

1. Message Element Setter mediation primitive
2. Data Handler mediation primitive

Both of these mediation primitives can change a subsection within the context, the headers, the body, or the attachments of the SMO. The Message Element Setter mediation primitive sets, deletes, or copies content into specified fields, or appends to an array of fields within a message. The Data Handler mediation primitive transforms native data to a business object, or vice-versa, so that it has a structure that can be accessed within the mediation flow component.

Message Element Setter mediation primitive

The Message Element Setter mediation primitive provides a simple mechanism for setting message content. It does not change the type of the message. You can set, copy, append or delete message elements using statements containing XPath expressions. You can specify multiple statements in one Message Element Setter mediation primitive by setting multiple targets, with the changes carried out in the sequence specified.

You can set target message elements to a constant value, or to a value copied from a location in the input SMO. If the target message element you specify does not exist then it will be created in the SMO. You can also delete message elements if they are optional or repeating elements.

Example scenario

The fictional company Personal Holidays are making an internal service available to external clients using a Web Service interface and a mediation flow, as shown in Figure 171.

Figure 171. Message Element Setter in a mediation flow component

The external clients send in SOAP messages, through the web service export, that contain a SOAP header called **PersonalHolidaysClient**. The value of this header must be stored in the SMO correlation context so it can be used in the response flow but should not be passed to the internal service. Additionally the internal service must have a SOAP Header named **PersonalHolidaysInternalHeader**, which must be set to a value of **EXTERNAL_CLIENT_SERVICE_CALL** for all incoming messages from the external clients.

We can use a Message Element Setter primitive to achieve this by:

1. Copying the value of the **PersonalHolidaysClient** SOAP header to the appropriate position in the correlation context.

2. Override the **PersonalHolidaysClient** to be **PersonalHolidaysInternalHeader** by:
   a. Setting the name of the **PersonalHolidaysClient** SOAP header to **PersonalHolidaysInternalHeader**.
   b. Setting the value of the **PersonalHolidaysInternalHeader** SOAP header to **EXTERNAL_CLIENT_SERVICE_CALL**.

Figure 172 shows the properties view within Integration Designer, where the actions of the Message Element Setter primitive are set. Each statement has the additional information needed to perform the action. All of the statements have a target XPath expression identifying the target of the action.

Figure 172. Message Element Setter properties within Integration Designer

Data Handler mediation primitive

The Data Handler mediation primitive transforms a targeted section of a message. The Data Handler primitive is used to convert an element of a message from a physical format to a logical structure, or from a logical structure to a physical format. The source and target of the transformation are indicated by XPath expressions. The Data Handler mediation primitive can be used when you are trying to integrate services, and parts of the message have not been parsed. By transforming the native data in a message into a business object, that data can be used by other mediation primitives. The part of the message that is not referenced by the XPath expression is left unmodified. Figure 173 on page 629 shows the data formats that can be transformed by a Data Handler mediation primitive. Additionally you can define and use your own data format transformation.

Figure 173. Data Handler Configuration

Example scenario

The fictional company My Airways outsource the supply of their airline meals to a fictional company called Flight Meals. My Airways specify that the ingredients of meals must be supplied to customers, should they request them. Flight Meals provide menuCourseIngredients information in a CSV format, as shown in the following XML:

```
<menuCourseDetails>
 <menuCourseID>DessertABC1234</menuCourseID>
 <menuCourseDescription>Sticky Date Pudding served with Caramel Sauce</menuCourseDescription>
 <menuCourseIngredients>Dates,Butter,Sugar,Bicarbonate of Soda,Eggs,Flour,
   Cream,Vanilla essence</menuCourseIngredients>
</menuCourseDetails>
```

My Airways process the menuCourseIngredients information into a business object format as shown in the following XML:

```
<menuCourseDetails>
 <menuCourseID>DessertABC1234</menuCourseID>
 <menuCourseDescription>Sticky Date Pudding served with Caramel Sauce</menuCourseDescription>
 <menuCourseIngredients>Dates,Butter,Sugar,Bicarbonate of Soda,Eggs,Flour,
   Cream,Vanilla essence</menuCourseIngredients>
 <ingredientsList>
  <ingredients>Dates</ingredients>
  <ingredients>Butter</ingredients>
  <ingredients>Sugar</ingredients>
  <ingredients>Bicarbonate of Soda</ingredients>
  <ingredients>Eggs</ingredients>
  <ingredients>Flour</ingredients>
  <ingredients>Cream</ingredients>
  <ingredients>Vanilla essence</ingredients>
 </ingredientsList>
</menuCourseDetails>
```

A Data Handler mediation primitive is used to transform data from a CSV format into the required business object. Figure 174 shows how a Service Invoke mediation primitive is used to call the Flight Meals Service Provider, to retrieve the menuCourseIngredients, containing the CSV formatted ingredients information from the Flight Meals Service Provider. The CSV formatted message is then passed to a Data Handler mediation primitive, which is able to transform the CSV data into a business object that can be used by the My Airways Service Provider.

Figure 174. Data Handler within a mediation flow component

Within Integration Designer, the source and target XPath expressions are specified for the Data Handler mediation primitive, and the **CSVDataHandler**, data format transformation is chosen. This is shown in Figure 175.

Figure 175. Data Handler in Integration Designer

WebSphere Transformation Extender

WebSphere Transformation Extender (WebSphere TX) is a separately licensed product and is not included as part of WebSphere ESB. Using WebSphere TX, you can create maps that can transform data between a wide variety of formats.

WebSphere TX for Integration Servers adds and enables the following components for use in your message flows:

- A mediation primitive for running WebSphere TX maps, called the WTX Map primitive
- A data format transformation for running WebSphere TX maps called the WTX Data Format Transformation

For more information on using WebSphere TX, see http://publib.boulder.ibm.com/infocenter/wtxdoc/v8r3m0/index.jsp.

There are some circumstances in which it is not possible to transform the message to your required structure using the BO Mapper, XSLT and Data Handler mediation primitives. In these cases WebSphere TX maps can be used to transform the data. WebSphere TX maps can be called using the WTX Map mediation primitive or by using the WTX Data Format Transformation in a Data Handler mediation primitive or in an Import or Export.

You can use the WebSphere TX maps and type trees in the following ways:

- Pulling data from sources such as files and queues within a map
- Calling external programs and services from a map
- Splitting and routing batches of messages

Use WebSphere TX type trees if you need message modeling features that are not available with native data format transformations, such as:

- Modelling complex binary or text message formats that Business Objects are unable to model
- Complex validation without having to write code, using component rules on type trees
- Use WebSphere Transformation Extender Industry Packs to integrate a range of industry standard data formats with your enterprise infrastructure through predefined type tree templates and maps that the packs provide.

Invocation of Services

Services can be invoked with WebSphere ESB, so that you can create solutions that interact with other services. You can use the Callout node or the Service Invoke mediation primitive to call a service. Service invocations can be made by both request and response flows, using synchronous and asynchronous programming styles. This sections details the different ways that services invocations can be made.

Service Invoke and Callout nodes

You can call a service using either a Service Invoke mediation primitive or a Callout node.

Service Invoke mediation primitive

A Service Invoke mediation primitive is used to call a service from inside a mediation flow, rather than waiting until the end of the mediation flow and using the Callout node. The input message is used to call the service, and if the call is

successful, the response is used to create the output message. If the call is unsuccessful, you can retry the same service or call another service.

You can have multiple Service Invoke mediation primitives inside one mediation flow. Therefore, you can have a series of service calls. You can use the Service Invoke mediation primitive in a request or response mediation flow.

Callout node

The Callout receives the message and calls the requested service and operation. There is a Callout node for each connected target operation in the mediation flow.

If the call is successful, the Callout Response node in the response flow receives the response message.

If the call is unsuccessful, the Callout can be set to retry service invocations depending on the type of fault received.

Comparison of the Service Invoke mediation primitive and the Callout node:

You can call a service using either a Service Invoke mediation primitive or a Callout node. You must consider the similarities and differences between using either the Service Invoke mediation primitive and the Callout node.

Similarities between the Service Invoke mediation primitive and the Callout node:

- The in terminal of the Service Invoke mediation primitive corresponds to the in terminal of the Callout node.
- The out terminal of the Service Invoke mediation primitive corresponds to the out terminal of the Callout Response node.
- The fail terminal of the Service Invoke mediation primitive corresponds to the fail terminal of the Callout Response node (for an unmodeled fault).
- A modelled fault output terminal of the Service Invoke mediation primitive, corresponds to a modelled fault terminal of the Callout Fault node.

Differences between the Service Invoke mediation primitive and the Callout node:

- The Service Invoke mediation primitive does not switch from request flow to response flow.
- The Service Invoke mediation primitive does not modify either the transient context or the correlation context.
- A Service Invoke mediation primitive has a separate timeout terminal. A Callout node only has a fail terminal.

Although, there is a functional similarity between the Service Invoke mediation primitive and the Callout node, the Service Invoke mediation primitive does not replace the Callout node. Consider the following when deciding whether to use the Service Invoke mediation primitive or the Callout node:

Table 57. Considerations for using the Service Invoke mediation primitive or the Callout node

Use the Service Invoke if	Use the Callout if
You need to call a service and return the response within a mediation flow.	You need to mediate a message (without calling an intermediate service) and then call a service provider. The Callout provides the simplest model for this configuration.
You need to interact with multiple services, and produce an output that combines service responses.	

Table 57. Considerations for using the Service Invoke mediation primitive or the Callout node (continued)

Use the Service Invoke if	Use the Callout if
You need to call an intermediate service. For example, you might use an intermediate service to adjust a message, or validate a message externally. The mediation flow contains a Service Invoke mediation primitive, and a Callout node that is connected to the service provider.	

Retry:

Callout nodes and Service Invoke mediation primitives provide retry capabilities for failed service invocations. A number of properties need to be set in order to enable and configure the retry.

- **Retry On**, determines whether, and how, fault responses cause a retry. The following values are valid:
  - Never - operation call is never retried, no matter the type of fault
  - Any fault - operation call is retried on both modelled and unmodeled faults
  - Modelled fault - operation call is retried only when a modelled fault is received
  - Unmodeled fault - operation call is retried only when an unmodeled fault is received

  Modelled faults are those that are explicitly listed in a WSDL file, any other fault is an unmodeled fault. This property is only applicable to request-response operations.
- **Retry Count**, determines how many times a service call should be retried before an output terminal is fired. The output terminal that is fired can be of the following types: modelled fault, timeout or fail. If `Try Alternate Endpoints` is false or if there are no alternate endpoints specified in the SMO headers, all attempts will retry the original service.
- **Retry Delay** is the number of seconds to wait between retry attempts. If left equal to `0`, there will be no delay between retry attempts meaning they will run as fast as the server can process them.
- **Try Alternate Endpoints**. If the SMO Header contains alternate endpoints, retry calls will be made to them in the order they appear. `Use Dynamic Endpoint` must also be set for this function to be available.
- **Async Timeout** specifies the time to wait for responses when performing an asynchronous with deferred response invocation. If this timeout occurs and the `Retry Count` has not been reached, it is treated as an unmodeled fault and retry will be performed. The retry delay will take place following the Async Timeout. If the `Retry Count` has been reached, then the Timeout terminal of the Service Invoke primitive, or the fail terminal of the Callout Response node is fired.

Service invocation styles

Using WebSphere ESB, service invocations can be made by both request and response flows.

With SCA, you can call service components using synchronous and asynchronous programming styles.

A component can be called synchronously or asynchronously, independent of whether the implementation is synchronous or asynchronous. This is shown in

Figure 176. The component interfaces are defined in the synchronous form and asynchronous support is generated for them. You can specify the preferred interaction style of the component as synchronous or asynchronous. The asynchronous style is used if interface contains at least one operation that can take a significant amount of time to complete. As a consequence, the service requester must avoid keeping a transaction active while waiting for the operation to complete. The interaction style applies to all the operations in the interface.

The underlying SCA programming model provides two different invocation styles that can be used for request-response operations:

1. Synchronous SCA invocation
2. Asynchronous SCA invocation
   - Asynchronous with deferred response SCA invocation
   - Asynchronous with callback SCA invocation

Figure 176. WebSphere ESB shows how a component can be called synchronously or asynchronously

Synchronous SCA invocation:

When a service component is called synchronously, or performs a synchronous invocation, both the service requester and the service provider run in the same thread (a thread of execution can be defined as the smallest unit of processing that can be scheduled by an operating system). The calling component within WebSphere ESB is blocked until a response is received from the provider.

Once the service provider has been called, all processing is suspended until the thread returns a response. This is shown in Figure 177.

Synchronous SCA invocation is useful if the service requester depends upon receiving a response from the service before continuing its processing.

Figure 177. Synchronous SCA invocation

Example scenario

Jane wants to transfer money through online banking, to Stephen's bank account. The bank must ensure that Jane's account is debited, and Stephen's account is credited. Both of the actions must be completed, and if one of the actions fail then neither of them must occur. If Jane's bank account is debited, and Stephen's bank account fails to be credited, a rollback action must occur to credit Jane's bank account, so that money is not lost at any point of the transaction. A confirmation message will only be sent to Jane if the transfer is successful.

Synchronous SCA invocation is an applicable style in this situation, because a message is sent to the service in one thread of execution, and no processing is done by the service requester until the message returns for processing to continue in the same thread.

You can complete this scenario using the user interface. If you are using the SCA programming model directly, then you can use the following Java code, which shows how this scenario is made possible in the implementation:

```
public void transferFunds(String from, String to, float amount) {

  BOFactory bofactory = (BOFactory)ServiceManager.INSTANCE.
    locateService("com.ibm.websphere.bo.BOFactory");

  DataObject creditDetails = bofactory.
    create("http://ServiceInvocationStyles", "Credit");
  creditDetails.setString("creditor", to);
  creditDetails.setFloat("amount", amount);

  Boolean creditSuccess = (Boolean)locateService_AccountInterfacePartner().
    invoke("creditAccount", creditDetails);
```

```
  //If the above synchronous bank credit fails then the debit will not occur
  if (creditSuccess.booleanValue()) {

   DataObject debitDetails = bofactory.
     create("http://ServiceInvocationStyles", "Debit");
   debitDetails.setString("debitor", from);
   debitDetails.setFloat("amount", amount);

   Boolean debitSuccess = (Boolean)locateService_AccountInterfacePartner().
     invoke("debitAccount", debitDetails);

   //Only when the complete transaction is complete do we send a confirmation
   if (debitSuccess.booleanValue()) {
    locateService_CustomerInterfacePartner().
      invoke("sendConfirmation", "Your transfer was successful");
   }
  }

 }
```

Asynchronous invocation:

When a service component is called asynchronously, the service requester and the service provider run in different threads. For an asynchronous invocation, the calling component within WebSphere ESB is not blocked while the request is processed by the provider. The SCA runtime returns control to the requester immediately, allowing it to continue processing.

With asynchronous invocation in SCA, there are three types of asynchronous interaction styles available: one way, deferred response, and request with callback. With all three types of asynchronous invocation, the service requester receives control back immediately from the SCA runtime.

There are two ways that the service requester can capture the response at a later time.

- **Asynchronous with deferred response**, where the service requester calls an operation on the service provider, and then continues processing. At a later time, the service requester calls the service provider for a response.
- **Asynchronous with callback**, where the service requester calls an operation on the service provider and specifies a callback handler. When the response is available, the SCA runtime returns the response to the requester by calling this callback handler.

Asynchronous with deferred response SCA invocation

Asynchronous with deferred response SCA invocation is useful if the service requester has further processing to perform that does not depend on information to be returned in the response.

When using asynchronous with deferred response invocation, resources within the SCA runtime thread are held until a response is available from the service provider. If the service provider takes a hours or days to process the request, the invocation style asynchronous with callback should be used. Asynchronous with deferred response SCA invocation is shown in Figure 178 on page 637.

Figure 178. Asynchronous with deferred response SCA invocation

Example scenario

A car insurance comparison website would prefer to use the asynchronous with deferred response style of invocation. The service requester requests a search on car insurance from two different car insurance providers. The responses are not dependent upon each other. When the car insurance providers return a response, the service requester can choose the cheapest quote. Figure 179 shows how the service requester can call both service provider A and B.

Figure 179. Car insurance comparison website using asynchronous with deferred response invocation

The service requester creates a request to service provider A, and calls the SCA runtime. The SCA runtime returns control to the requester immediately, allowing it to continue processing. The service requester is then able to create a second request for service provider B.

The service requester makes further calls to the SCA runtime to collect the responses. These are processed, and the final response is then returned to the user.

You can complete this scenario using the user interface. If you are using the SCA programming model directly, then you can use the following Java code, which shows how this scenario is made possible in the implementation:

```
  public float getBestQuote() {

  BOFactory bofactory = (BOFactory)ServiceManager.INSTANCE.
     locateService("com.ibm.websphere.bo.BOFactory");
  DataObject transferDetails = bofactory.
    create("http://ServiceInvocationStyles", "CarDetails");
  transferDetails.setString("make", "Ford");
  transferDetails.setString("model", "Focus");

  //Send the first request
  Ticket quote1Ticket = locateService_InsuranceQuotePartner().
    invokeAsync("getQuote", transferDetails);

  //Send the second request
  Ticket quote2Ticket = locateService_AnotherInsuranceQuotePartner().
    invokeAsync("getQuote", transferDetails);

  long waitFor = 10000;

  //Get the response from the first service
  Float quote1 = (Float)locateService_InsuranceQuotePartner().
    invokeResponse(quote1Ticket, waitFor);

  //Get the response from the second service
  Float quote2 = (Float)locateService_AnotherInsuranceQuotePartner().
   invokeResponse(quote2Ticket, waitFor);

  if (quote1 < quote2) {
   return quote1;
  } else {
   return quote2;
  }
 }
```

Asynchronous with callback SCA invocation

In this invocation style, the calling component (service requester) specifies a callback handler that will be used to receive and process the response. After the service requester has sent the request, the thread that made the request completes processing. SCA runtime resources associated with that thread are then released. When the response from the service provider is available, a new SCA runtime thread is used to call the callback handler to perform the processing required by the service requester.

This style of invocation is used for long running processes or services which could take hours or days to return a response. Asynchronous with callback SCA invocation is shown in Figure 180 on page 639.

Figure 180. Asynchronous with callback SCA invocation

Example Scenario

A company would prefer to use this style of invocation if they perform audits. The service requester wants to update employees' payment details, and only needs to be notified when the update has been made.

The service requester sends a series of calls to update employees' payment details, to the service provider. The service provider invokes an audit service for each request. The audit can take from a few hours to a whole day to make a record of each employees payment details. When the audit service is complete, the response is returned to the service requester using a callback handler, and an e-mail is sent to the user to notify them that the update has been successful.

You can complete this scenario using the user interface. If you are using the SCA programming model directly, then you can use the following Java code, which shows how this scenario is made possible in the implementation:

```
  public void updateEmployees(List<Employee> employees) {
  BOFactory bofactory = (BOFactory)ServiceManager.INSTANCE.
    locateService("com.ibm.websphere.bo.BOFactory");


   Iterator <Employee> iterator = employees.iterator();
   while(iterator.hasNext()){
   Employee employee = (Employee) iterator.next();

   DataObject employeeDetails = bofactory.
     create("http://ServiceInvocationStyles", "Employee");
   employeeDetails.setString("firstName", employee.firstName);
   employeeDetails.setString("lastName", employee.lastName);
   employeeDetails.setInt("lastName", employee.age);

   locateService_EmployeeInterfacePartner().
     invokeAsyncWithCallback("updateEmployee", employeeDetails);
  }
 }
 /**
  * Method generated to support the async implementation using callback
  * for the operation "EmployeeInterface#updateEmployeeDetails(DataObject employee)"
  * of wsdl interface "EmployeeInterface"
  */
 public void onUpdateEmployeeDetailsResponse(Ticket __ticket,
   DataObject returnValue, Exception exception) {

  BOFactory bofactory = (BOFactory)ServiceManager.INSTANCE.
    locateService("com.ibm.websphere.bo.BOFactory");
  DataObject email = bofactory.create("http://ServiceInvocationStyles", "Email");
```

```
  email.setString("to", returnValue.getString("email"));
  email.setString("subject", "Employee record updated");
  email.setString("content",
    "Your employee details have been successfully updated to "+returnValue);

  locateService_EmployeeInterfacePartner().invoke("sendEmail", email);
 }

 public class Employee {
  String firstName;
  String lastName;
  int age;
 }
```

Invocation from within the mediation flow component:

The configuration of the Service Invoke mediation primitive or the Callout node can determine the invocation style that is used.

Within Integration Designer, the options for the invocation style configuration are:

- Sync
- Async
- Default

Sync invocation style

The synchronous invocation style can be used in all cases, regardless of whether the operation is one-way or request-response, and regardless of the invocation style used to invoke the mediation flow component.

Async invocation style

When the invocation style option is set to async, the invocation style used by the mediation flow component for service invocations is determined by the following options:

Table 58. Invocation style used by the mediation flow component for service invocations

How the mediation flow component is called	OneWay/ Request-Response	Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback	Invocation style
Synchronous	Both	Any	Asynchronous
Asynchronous	Both	Any	Asynchronous
Asynchronous with callback	OneWay	Any	Asynchronous
	Request-Response	True	Asynchronous
		False	Asynchronous with callback

The Service Invoke mediation primitive and the Callout node both have a property that, when specified, prevents the mediation flow component from using the asynchronous with callback style for the service invocation. The Integration Designer tool identifies this property with a check box labeled **Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback**.

Default invocation style

The default option is available, to support legacy applications.

When the invocation style is default, the invocation style used by the mediation flow component for service invocations is determined by three factors:

- The preferred interaction style of the target import or component.
- The invocation style used to call the mediation flow component.
- The use of the property: **Require mediation flow to wait for service response when the flow component is called asynchronously with callback.**

For more information about the Default invocation style, see: http://www.ibm.com/developerworks/library/ws-websphereesb2/.

Dynamic invocation

When a mediation module is developed and deployed, the flow of messages through the module is determined by the wiring of components. The configuration of the components is used to determine which external services are to be called.

Applications can override or change the configuration and reroute messages at run time. You might do this dynamically by overriding a value specified for an endpoint address. Alternatively, you can select a new target import. In each case, the message flow changes according to the information in the message.

For example, you can use Integration Designer to create a binding on an import. The import contains endpoint information, specifying the location of a remote service. This fixed endpoint information can be overridden dynamically by information carried in the message.

If a message flow is changed dynamically within a mediation module, the new route applies only to that message and any response coming back. When the message has been sent and any response returned, the dynamic routing changes are discarded and the original fixed values defined in the module are used as the default values again. Any response to a dynamic invocation returns using the same route as the dynamic invocation.

There are three main ways that dynamic invocation takes place:

- Dynamic override of a fixed endpoint, in which a service is called using any supported import binding. The service is available at a different endpoint from the endpoints originally specified when the mediation module was created and deployed.
- Dynamic invocation with a target import, in which a service is called using any supported import binding. The binding must already be available within the mediation module. It is selected at run time according to information contained in the message.
- Pure dynamic invocation, in which a service is called without needing an import in the mediation module. No additional information is required apart from the details in the message.

The Endpoint Reference (EPR) structure within the request message is used to provide the information required for a dynamic invocation. Details about the EPR can be obtained by retrieving information about the endpoint from a suitable storage location, such as WSRR or a database. This information is used to set EPR details for the message using the SCA Endpoint Reference API. The SMO structure:

```
/headers/SMOHeader/Target/address
/headers/SMOHeader/Target/@bindingType
/headers/SMOHeader/Target/@import
/headers/SMOHeader/AlternateTarget/address
/headers/SMOHeader/AlternateTarget/@bindingType
/headers/SMOHeader/AlternateTarget/@import
```

During a dynamic invocation, the invocation target type is identified by examining the endpoint URI.

- If the URI begins with `sca`, this indicates an SCA component.
- A URI prefix of `http` or `jms` is assumed by default to indicate a web service endpoint.
- A URI prefix of `http` does not by default indicate an HTTP service, and a URI prefix of `jms` does not by default indicate a JMS service. However, if the reference is wired to an import, or a target import is specified with a JMS or an HTTP binding, the URI is assumed to be JMS or HTTP, and not web services.
  - The `bindingType` field in the Endpoint Reference can be used to change this. For example, using the SCA Endpoint Reference API to set the `bindingType` field to a value of `EndpointReference.BINDING_TYPE_HTTP` or a mediation flow to set the `Target@bindingType` attribute to HTTP, results in a URI prefix of `http` in the EPR being interpreted as indicating an HTTP binding for that message.

Dynamic override of a fixed endpoint:

Dynamic invocation by overriding a fixed endpoint can take place, when a service is called using any supported import binding.

This capability is used when the service is available from a different endpoint than originally specified when the mediation module was created and deployed.

Dynamic override of a fixed endpoint can be considered if a company is carrying out maintenance on the fixed service. In this instance, the company does not need to change their entire mediation flow to accommodate for a back up service. The company must modify the target address in the SMO header, which is done by registering the Back-up Service in WebSphere Service Registry and Repository (WSRR) and using an Endpoint Lookup mediation primitive to retrieve the Back-up Service target address details.

Figure 181 on page 643 shows how a message received by this module would usually be sent to the fixed endpoint for the Internal Service. The SMO can be modified to override the fixed endpoint and route the message to a Back-up Service.

A new target address is set for the message by changing SMO values, using the Endpoint Lookup mediation primitive, and registering the Back-up Service in WSRR. The message can then be dynamically routed to the Back-up Service.

Figure 181. Dynamic override of a fixed endpoint using an Endpoint Lookup mediation primitive

Dynamic invocation with a target import:

Dynamic invocation of endpoints can be enhanced using the `Target@import` attribute within the SMO header.

Dynamic invocation with a target import allows the configuration and protocol used to call a service to be dynamically selected at run time by identifying a supported import binding. The import and its binding must already be available within the module and is selected at run time according to information contained in the message.

You can use Integration Designer to create a mediation module that selects the target service dynamically at run time. The target services might use different protocols, different formats or different quality-of-service values. Each combination must be known at the time the mediation module is developed. This means that for each combination of protocol, format, and quality-of-service value, Integration Designer includes an import in the mediation module with an appropriate configuration.

For example a financial services company that provides an interactive web-based stock market service to its customers, has introduced tiered levels of service for Gold and Standard customers:

- Gold customers receive a reliable real-time stock quote, which is mapped within the mediation flow component using a JMS binding.
- Standard customers receive a delayed stock quote, which is mapped with a WebServices binding.

Figure 182. Dynamic invocation with a target import

As shown in Figure 182, two imports are needed in the mediation module for the two different bindings. During development of the mediation module, an import might be wired to another component or might be left unwired. If the module is unwired, a connection must be created at run time by specifying a target import and creating the connection dynamically.

In addition to dynamically selecting an import, a new target address can also be set using the incoming message content and routing information. The mediation flow component sets the new address by changing the SMO values using the Endpoint Lookup mediation primitive, and registering the Gold and Standard Services in WSRR. The choice between the target service destinations is made dynamically at run time using available metadata provided by the message or by WSRR. The name of the target import is placed in the SMO Header by the Endpoint Lookup mediation primitive. If an import is targeted that does not support the target endpoint, a ServiceRuntimeException will occur.

Pure dynamic invocation:

Pure dynamic invocation calls a service without needing an import in the mediation module.

No additional information is required apart from the details in the message. An example is where the intended service is identified by a name, which is looked up at runtime. Pure dynamic invocation assumes a specific set of configuration parameters.

Web services and SCA URLs are supported, but other types are not because of their need for additional configuration that is provided by an import binding.

The default for pure dynamic invocation using a Web Service binding is SOAP 1.1, using HTTP or JMS as specified in the target URL. Pure dynamic invocation is also supported for SOAP 1.2 when using HTTP by specifying an appropriate value for the bindingType field of the EPR. Pure dynamic invocation using SOAP 1.2 and JMS is not supported.

The default for pure dynamic invocation using a Web Service binding is SOAP 1.1, using HTTP or JMS as specified by the target URL. Pure dynamic invocation is also supported for SOAP 1.2 with HTTP.

- For pure dynamic invocation using SOAP 1.1 and HTTP or JMS, the binding type should either be unset or it should be set to one of the following values:
  - Using the SCA Endpoint Reference API:
    - EndpointReference.BINDING_TYPE_WEB_SERVICE
    - EndpointReference.BINDING_TYPE_WEB_SERVICE_SOAP_1_1
  - Using the SMO structure
    - Target@bindingType=WS
    - Target@bindingType=WS1.1
  -
- For pure dynamic invocation using SOAP 1.2 and HTTP, the EndpointReference binding type should be set to:
  - EndpointReference.BINDING_TYPE_WEB_SERVICE_SOAP_1_2 using the SCA Endpoint Reference API
  - Target@bindingType=WS1.2 using the SMO structure

Figure 183. Pure dynamic invocation

An example of when pure dynamic invocation could be used is with a search engine. Figure 183 shows how a call can be made to the mediation flow component. An Endpoint Lookup mediation primitive is used to retrieve all endpoints from a public WSRR. A public WSRR allows companies to register themselves with service providers on an ongoing basis. This indicates that the endpoint is constantly changing, and so a pure dynamic environment is required. The advantage of pure dynamic invocation, is that the mediation module does not need to be changed in order to respond to new service providers.

Dynamic endpoint selection:

In a mediation flow, you can select a service address based on a condition or property in your flow, and call an endpoint using the selected service. This section details some of the different mediation primitives that retrieve an endpoint address from a registry, to route the message to the retrieved endpoint.

Endpoint Lookup mediation primitive:

Use the Endpoint Lookup mediation primitive to dynamically route messages to appropriate service endpoints. The Endpoint Lookup searches for service information in a WSRR registry.

You can use the Endpoint Lookup mediation primitive to retrieve service endpoint information from a WSRR registry that can be local or remote. The service endpoint information can relate directly to web services, or to SCA module exports, or to manually defined MQ, JMS and HTTP services.

In order to use the Endpoint Lookup mediation primitive you need to add service endpoint information to your WSRR registry.

- If you want to extract service endpoint information about web services, your WSRR registry must contain either the appropriate WSDL documents, or SCA modules that contain exports using web service bindings.
- If you want to extract service endpoint information about exports that use the other SCA bindings, your WSRR registry must contain the appropriate SCA modules.
- If you want to extract service endpoint information about services that are accessed using MQ, JMS or HTTP but are not defined in a SCA module, your WSRR registry must contain an appropriate "Manual JMS/HTTP/MQ endpoint with associated Interface" Business Object with the endpoint information and associated interface correctly set.

The Endpoint Lookup mediation primitive lets you retrieve service endpoint information that relates to the following:

- Web services using SOAP/HTTP.
- Web services using SOAP/JMS.
- SCA module exports with Web Service bindings, using SOAP/HTTP.
- SCA module exports with Web Service bindings, using SOAP/JMS.
- SCA module exports with the default SCA binding.
- SCA module exports with the MQ binding.
- SCA module exports with the MQ JMS binding.
- SCA module exports with the JMS binding.
- SCA module exports with the Generic JMS binding.
- SCA module exports with the HTTP binding.
- Manually defined MQ endpoints with associated interfaces.
- Manually defined JMS endpoints with associated interfaces.
- Manually defined HTTP endpoints with associated interfaces.

When the Endpoint Lookup mediation primitive receives a message it sends a search query to the registry. The search query is constructed using the Endpoint Lookup properties that you specify and the query might return nothing, or it might return one or more service endpoints.

The Endpoint Lookup primitive will update the target structure in the SMO header with the first service endpoint. The Endpoint Lookup primitive will then update the alternative target list with the remaining service endpoints.

Example scenario

Figure 184. Endpoint Lookup within a mediation flow

The fictional company Personal Holidays is making an internal service available to external clients using a mediation flow. If the server that runs the internal service fails, one or more back-up servers which are also hosting the internal service are called instead, as shown in Figure 184.

The scenario is implemented by registering the service endpoints of the main server and the back-up servers in WSRR. Personal Holidays then use the Endpoint Lookup Primitive to retrieve all of the endpoints from WSRR. Figure 185 shows the Endpoint Lookup mediation primitive's properties.

Figure 185. Endpoint Lookup mediation primitive properties

Personal Holidays also set the **Try alternate endpoints** property on the Callout node to try the main server endpoint first. If that fails, the Callout node will retry the call using the service endpoint information for the back-up servers. The Callout node properties are shown in Figure 186 .

Figure 186. Try alternate endpoint property of the Callout node

By storing the service endpoints in WSRR, Personal Holidays can update their back-up services at any time, without needing to modify the mediation flow component.

UDDI Endpoint Lookup mediation primitive:

Use the UDDI Endpoint Lookup mediation primitive to retrieve service endpoint information from a UDDI version 3 registry. The UDDI registry only holds Web Services information.

To use the UDDI Endpoint Lookup mediation primitive you need to add service endpoint information to your registry.

The UDDI Endpoint Lookup mediation primitive lets you retrieve service endpoint information that relates to the following:

- Web Services using SOAP/HTTP.
- Web Services using SOAP/JMS.

When the UDDI Endpoint Lookup mediation primitive receives a message it sends a search query to the registry. The search query is constructed using the UDDI properties that you specify and the query might return nothing, or it might return one or more service endpoints.

The UDDI Endpoint Lookup primitive will update the target structure in the SMO Header with the first service endpoint. The UDDI Endpoint Lookup mediation primitive will then update the alternative target list with the remaining service endpoints.

The UDDI Endpoint Lookup "Example scenario" on page 646, can be implemented with a UDDI version 3 registry, and a UDDI Endpoint Lookup mediation primitive in place of WSRR.

Service Level Agreement Endpoint Lookup mediation primitive:

Use the Service Level Agreement (SLA) Endpoint Lookup mediation primitive to look up endpoints referenced from an active service level agreement (SLA) between a service consumer and a service provider.

If you use WebSphere ESB to integrate and model your business in WSRR, using its Governance Enablement Profile (GEP), you can select dynamic endpoints based on a number of factors modelled in that profile. For example:

- Whether the consumer of the endpoint has a valid SLA for the endpoint.
- Whether the particular SLA is active.
- Whether the endpoint is online.
- Whether the endpoint has a certain desired classification; for example, whether it is Production or Development.

Example scenario

The fictional company Personal Holidays is making an internal service available to external clients using a mediation flow. They want to use the WSRR GEP to ensure that only consumers with a valid SLA can access the internal service.

The scenario is implemented in a similar way to the "Example scenario" on page 646, using an SLA Endpoint Lookup mediation primitive in place of the Endpoint

Lookup mediation primitive. The SLA Endpoint Lookup primitive is configured to retrieve a consumer ID from the SMO and to relay that information to WSRR. WSRR returns endpoint information only for services that have an SLA and the SLA consumer matches the provided consumer ID, as shown in Figure 187.

Figure 187. SLA Endpoint Lookup

Custom Endpoint Lookup mediation primitive:

There are situations in which the supplied Endpoint Lookup mediation primitives are not sufficiently flexible. In these circumstances, use a Custom mediation primitive. If you need to access data in a WSRR registry, use the WebSphere ESB WSRR proxy.

Example scenario

The fictional company My Airways has registered some WSDL-defined service providers in a WSRR registry, and want to retrieve the service endpoint URL that is based on a user property, which is determined from the SMO at runtime. My Airways use a Custom mediation primitive to retrieve the service endpoint URL and set it in the SMO, so the service providers are called dynamically.

The Custom mediation primitive code completes three steps:

- It sends an XPath query to the WSRR registry to look up the provider service WSDL information.
- It processes the results returned from WSRR, retrieving the service endpoint URL.
- It sets the service endpoint URL into the SMOHeader Target address field. The URL is used when an import with a Web Services binding is called, and the Callout node has been configured to use a dynamic endpoint.

The following is the Custom mediation primitive code:

```
String endpointAddress = null;

// Get the factory and then get the default instance of a ServiceRegistryProxy.
Can also use getServiceRegistryProxy("RegistryName")
com.ibm.wsspi.sibx.mediation.wsrr.client.ServiceRegistryProxy srProxy =
com.ibm.wsspi.sibx.mediation.wsrr.client.ServiceRegistryProxyFactory.getInstance().
getDefaultServiceRegistryProxy();

try
{
   // Set up the WSRR XPath query - this query looks for WSDL Ports that
    have the PortType name 'MyAirwaysAccount'and the required customerType user property
   // and the PortType namespace 'http://www.MyAirways.com/AccountManagement/'
   String customerType = smo.getString("/body/bookFlight/customerType");
    String query = "/WSRR/WSDLService/ports[binding(.)/portType(.)
```

```
   [@name = 'MyAirwaysAccount' "+
       "and @namespace='http://www.MyAirways.com/AccountManagement/'
       and @customerType ='" + customerType + "']]";

   // Run the query & get the results
  com.ibm.wsspi.sibx.mediation.wsrr.client.data.
  ServiceRegistryDataGraphList list = srProxy.query(query, -1, false);

   // Iterate through the results
   Iterator<DataGraphType>  resultsIterator = list.getDataGraphs().iterator();
   while(resultsIterator.hasNext() && endpointAddress == null)
   {
     DataGraphType dgt = resultsIterator.next();
     BaseObject[] baseObjects = dgt.getWSRR().getArtefacts();


    com.ibm.ws.management.discovery.EndpointAddress endpointAddress = null;

     // Find the SOAP Address
    for(int i=0; i<baseObjects.length; i++)
    {
         BaseObject bo = baseObjects[i];
         if(bo instanceof com.ibm.wsspi.sibx.mediation.wsrr.client.jaxrpc.WSDLService)
         {
           com.ibm.wsspi.sibx.mediation.wsrr.client.jaxrpc.WSDLPort[] ports = ((com.ibm.
          wsspi.sibx.mediation.wsrr.client.jaxrpc.WSDLService)bo).getPorts();
            // Look for the SOAP Address in the first port

            endpointAddress = ports[0].getSOAPAddress().getLocation();
            break;
         }
      }
   }
}
catch(ServiceRegistryProxyException e)
{
   e.printStackTrace();
}

System.out.println(endpointAddress);

// If we have successfully retrieved an endpoint from WSRR, set it into the SMO
if(endpointAddress != null)
{
   smo.setString("headers/SMOHeader/Target/address", endpointAddress);
}
```

Service gateway

A service gateway is a single access point and acts as a proxy for multiple services. A service gateway enables transformations, routing, and common processing across all the services.

A service gateway module is a single mediation that handles the requests for multiple service consumers and providers. There are four steps that are typical of any service gateway:

1. **Common processing** - once the message is received by the gateway, common processing occurs for all messages, such as adding protocol level headers or logging the message.
2. **Service identification** - the message that is processed by the gateway needs to be identified as a certain service type. For example, the message is queried to determine if it is for service provider A, B, or C.
3. **Endpoint routing** - when it is known that a message will be delivered to a particular service provider, it is mapped to a network-addressable endpoint so that the message can be forwarded to the service provider.

4. **Service-specific processing** - any processing that is required for the particular target service is performed.

The order of the four steps can be changed according to different scenarios.

There are three service gateway patterns provided out of the box that are supported by WebSphere ESB:

1. "Dynamic service gateway pattern" on page 656
2. "Static service gateway pattern" on page 657
3. "Proxy gateway pattern" on page 658

Each of these patterns follow a general format within the tooling environment in IBM Integration Designer, as shown in Figure 188. The service gateway is represented as an SCA module. An export exposes the endpoint and has the service gateway interface attached. The service gateway interface instructs the runtime that it is in the service gateway mode.

Figure 188. General pattern for a service gateway module

The service gateway interface has two operations: requestOnly and requestResponse. Both operations use messages that have an anyType structure. An anyType message structure allows it to be one of five core structures that represent raw data that is sent to the mediation flow component, as shown in Figure 189 on page 652. This is not to accommodate any business object. Implement one or both of the interface operations.

Figure 189. Five structures that represent raw data

The binding type must be taken into consideration, to determine which business object is created. The following table shows the business objects that are created for the different binding types.

Binding type	**Business object created**
WebSevice Binding	TextBody
HTTP Binding	
HTTP Content-Type=text/*	TextBody
HTTP Content-Type=application/soap+xml	TextBody
HTTP Other	BytesBody
JMS Binding	
JMS TextMessage	TextBody
JMS BytesMessage	BytesBody
JMS StreamMessage	StreamBody
JMS MapMessage	MapBody
JMS ObjectMessage	ObjectBody
MQ Binding	
MQ Format = MQSTR	TextBody
Other	BytesBody

The content of the message is in a serialized form (such as XML) when it reaches the mediation flow component and is held within a TextBody structure. Header information is parsed as a business object and can be found in the header section of the Service Message Object (SMO). To change the body of the message, it can be parsed within the mediation flow component and changed using transformation mediation primitives and then serialized before it is passed to the service gateway import. To parse the message within the mediation flow component, the schema information associated with the message type should be available to the module.

There are two ways that WebSphere ESB can parse a message within the service gateway pattern:

1. "Manual parsing of the gateway message" on page 653
2. "Automatic parsing and any message type support" on page 653

Manual parsing of the gateway message

The DataHandler mediation primitive can be used to convert a message from a serialized form (such as XML) to a business object structure. This mediation primitive differs from other transformation primitives such as XSLT or the BO Mapper, which convert one business object into another business object.

The DataHandler mediation primitive is used extensively within the static service gateway pattern, where you must parse the serialized message body into the concrete business object structure. Each DataHandler primitive can only handle a single message type, therefore branching and multiple DataHandler primitives are required, as shown in Figure 190.

Figure 190. DataHandler mediation primitive used in a service gateway

Automatic parsing and any message type support

In IBM Integration Designer, as a default all terminals are defined with concrete message types. The terminals can be defined to have any message structure, which is set on individual terminals, allowing any business object to move through the mediation flow. WebSphere ESB parses the serialized message body into a concrete business object at the edge of the mediation flow component (input and callout). To enable this, the input nodes must be configured with the feature titled **Automatically convert the service gateway message**. This allows the parsing and serialization to occur automatically when the inbound message is XML encoded within a TextBody structure. The overall service gateway design is shown in Figure 191 on page 654.

Figure 191. Automatic parsing used in a service gateway

Patterns Explorer

IBM Integration Designer provides patterns that accelerate the development of a service gateway. Each pattern includes a wizard that generates a service gateway implementation, which is based on a set of parameters you provide. You can make updates to the generated implementation to complete your service gateway.

Patterns can be accessed through IBM Integration Designer Patterns Explorer, where you can create an integration solution for common scenarios. The following service gateway patterns are available:

- Proxy gateway pattern
- Static gateway pattern
- Dynamic service gateway

Figure 192. Pattern Explorer in IBM Integration Designer

Once a pattern has been selected, a pattern specification view opens which provides documentation about the pattern and its use. Click the 'Create New Instance' button, which opens a pattern configuration view. Specify the parameters

used to generate the pattern, as shown in Figure 193.

Figure 193. Pattern Explorer configuration view

Once the parameters have been entered, click the Generate button which creates the SCA module artifacts and provides a summary of information about the generated module as shown in Figure 194 on page 656.

Figure 194. Pattern Explorer summary information

Dynamic service gateway pattern

With the dynamic service gateway, you can enter new services into a gateway without any modification to the deployed module.

Schema information for services that are handled by the gateway are optional, if access to the body of the message is not required. This provides the flexibility to build a customizable service gateway, adding and removing services as required based on an external registry or database for endpoint resolution. Figure 195 on page 657 shows the common pattern used for a dynamic service gateway. The dynamic service gateway can be used with either web services, JMS, HTTP or MQ bindings.

Figure 195. Common pattern that is used for a dynamic service gateway

To create a web service client, you retrieve the WSDL file for the service provider, and pass it into the web service tooling to generate a client stub. For a dynamic service gateway the creation of this WSDL file is a manual process, where you must download the target service provider WSDL, replace any policy information with that attached to the service gateway export and then override the endpoint to that of the service gateway.

Example scenario:

A fictitious airline company has four different services that are based on different protocols:

- Flight booking - MQ binding
- Seat reservation - JMS binding
- Flight schedule - HTTP binding
- Food selection- web services binding

The airline company have a requirement that all messages must be audited in a single unit before the message reaches the service provider. In the future the airline company want to be able to associate new services without redeploying the gateway. The dynamic service gateway is used because it has a multi protocol requirement.

Static service gateway pattern

The static service gateway provides a single access point for many service requesters and service providers.

The available service types are static and cannot be changed after the module has been deployed to the runtime. It is possible to override the endpoint that the service exposes by using a service registry or database. The benefit of using this gateway pattern, is that the message body can be changed easily within the module. The static service gateway pattern can be used with either HTTP, JMS, MQ or web services. Figure 196 on page 658 shows the common pattern used for a static service gateway.

Figure 196. Static service gateway pattern

To create a web service client, you retrieve the WSDL file for the service provider, and pass it into the web service tooling to generate a client stub. For a static service gateway the creation of this WSDL file is a manual process, where you must download the target service provider WSDL, replace any policy information with that attached to the service gateway export and then override the endpoint to that of the service gateway.

Example scenario:

A fictitious car insurance comparison web site provides quotes to customers from 3 different car insurance providers. Each car insurance provider service has a different security configuration.

The comparison web site must audit the body of the message before it is forwarded on to the relevant service providers. The static service gateway is used because, only a section of the message is audited. Since each service provider has a different security configuration, a different SCA import is needed for each service provider, which is provided using the static service gateway pattern.

Proxy gateway pattern

The proxy gateway provides a generic capability across web service only service providers. This pattern has no dependencies on other products, such as a registry.

Figure 197. Proxy gateway

Figure 197 shows the relationship between a proxy gateway, proxy group, service provider and virtual service.

- Virtual service - the exposed endpoint of the proxy gateway for a service provider within the associated proxy group.
- Proxy gateway - the entity that provides the gateway function for one or more proxy groups.
- Proxy group - a collection of service providers that are grouped together into a logical set.

Configuration store:

WebSphere ESB includes a built-in configuration store which holds data about the relationships between the proxy gateway, proxy groups and associated service providers.

To change the data within the built-in configuration store, a web based interface is provided. Using the interface you can create, delete and amend virtual services associated with a proxy group.

Figure 198. High level interactions of the proxy gateway, the configuration store and the web based interface

Figure 198 shows the high level interactions of the proxy gateway, the configuration store and the web based interface. The "Built-in Configuration Store" maintains the associations between proxy groups and proxy gateways. The proxy gateways contain definitions of virtual services and their defined endpoints. The browser window shows the Business Space widget that is used to administer the proxy groups and their virtual service definitions.

Virtual services:
Network addressable endpoints for virtual services

The proxy gateway has an individual network addressable endpoint for each virtual service it is exposing, for example:

```
http://localhost:9080/proxyGateway/sca/ProxyGatewayExport/VirtualServiceA
http://localhost:9080/proxyGateway/sca/ProxyGatewayExport/VirtualServiceB
...
```

The proxy gateway web service endpoint accepts messages based on a URL pattern instead of the typical single URL pattern of a normal web service export.

Normal web service export URL:

```
http://localhost:9080/proxyGateway/sca/ProxyGatewayExport
```

Proxy gateway web service export URL pattern:

```
http://localhost:9080/proxyGateway/sca/ProxyGatewayExport/*
```

Virtual service WSDL generation

To create a web service client, you would normally retrieve the WSDL file for the service provider, and pass it into the web service tooling to generate a client stub. It might be difficult to retrieve the WSDL from a service gateway because the WSDL that the client uses is a combination of the WSDL for the gateway and the WSDL for the target service provider. For a dynamic or static service gateway the creation of this WSDL file is a manual process: you must download the target service provider WSDL, replace any policy information with the WSDL attached to the service gateway export and then override the endpoint to that of the service gateway. This lets the service consumer know that a service gateway is being used, and because a manual process was used to edit it, there is a chance that mistakes have been made when it was edited.

Automated WSDL resolution is supplied for the proxy service gateway, based on the following pattern:

```
<NORMAL-EXPORT-URL>/<VIRTUAL-SERVICE-NAME>?wsdl
```

For example:

```
http://localhost:9080/proxyGateway/sca/ProxyGatewayExport/VirtualServiceA?wsdl
```

Virtual service endpoint resolution

Since the gateway is a proxy for multiple service providers, it is necessary to be able to determine the type of message the gateway has received, and resolve this to a network addressable endpoint. The proxy gateway handles this automatically using the dedicated URL and the information in the built-in configuration store. The virtual service name is extracted ffrom the URL pattern and passed to the built-in configuration store to resolve it to a network addressable endpoint.

Introduction to Business Space:

Business Space provides a browser-based graphical user interface that lets users interact with content from products in the IBM Business Process Manager portfolio. It provides a customizable and collaborative environment for monitoring, reviewing, and administering common business processes, such as human task flows, mediations and modeling.

Business Space includes the concept of spaces, templates, pages and widgets. A space is a collection of pages based on a template, and a page is a canvas that contains a number of widgets. Widgets are the pluggable user interface

components that you use to define the functionality of your business spaces. WebSphere ESB and IBM Business Process Manager ship a number of built-in templates and widgets.

In the context of WebSphere ESB, Business Space provides administrative capabilities. You can control and manage the applications installed on the system using these capabilities.

Proxy gateway widget:

The proxy gateway widget displays a list of proxy groups that exist in the built-in configuration store.

Each proxy group in the list includes information about the installed proxy gateways that are associated with the group. When a proxy gateway is installed on a server, the proxy groups it references are automatically added to the configuration store if they don't already exist. If you uninstall a proxy gateway from the server, the proxy groups it is associated with are not deleted. This prevents the unintentional loss of configuration data for the proxy group when no proxy gateway applications are installed that reference it. You must delete the proxy group if it is no longer needed.

Figure 199 shows the proxy gateway widget in Business Space. It shows what you see when:

- a proxy group is shared by multiple proxy gateways
- a proxy gateway has multiple proxy groups
- a proxy group exists without an associated proxy gateway

Figure 199. Proxy gateway widget in Business Space

Figure 199 shows:

- There are two proxy gateways: PGW1_Gateway and PGW2_Gateway.
- There are four proxy groups: PGW1ProxyGroup, PGW2ProxyGroup, PGW3ProxyGroup, PGWSharedGroup.
- The first row has a proxy group (PGW1ProxyGroup) that is associated with a single proxy gateway (PGW1_Gateway).
- The second row has a proxy group (PGWSharedGroup) shared by multiple proxy gateways (PGW1_Gateway, PGW2_Gateway).
- Rows two and three illustrate a proxy gateway (PGW2_Gateway) that is associated with multiple proxy groups (PGWSharedGroup, PGW2ProxyGroup).

- The fourth row has a proxy group (PGW3ProxyGroup) without an associated proxy gateway.

Using the proxy gateway widget

Figure 200. Select a proxy gateway widget

Figure 200 shows how to select a proxy gateway to edit. To do this you must:

1. Select the relevant proxy group row in the table, which displays a pencil icon.
2. Click the (pencil icon) which changes the widget to a new view where you can configure the virtual services for the proxy group.
3. If the selected row is for a proxy group without an associated proxy gateway, a (X icon) is visible where you can delete the proxy group.

When you create a new proxy group in the configuration store, there are no virtual services defined. When you edit it for the first time, the panel shown in Figure 201 is displayed, where you must add a virtual service by filling out the WSDL Location field.

Figure 201. Editing a new proxy group

To add a virtual service, you must enter the URL to the WSDL that is usually available by entering the service endpoint, followed by ?wsdl. For example:

```
http://localhost:9080/BankServiceProviderWeb/sca/accountExport1?wsdl
```

(If the service provider does not support the ?wsdl convention, the WSDL can be loaded from another location such as a registry.)

You must then click Add Service, which opens the panel for defining a virtual service, as shown in Figure 202 on page 664.

Figure 202. Editing a new proxy group

The information in Figure 202 can be separated into six sections:

1. Port Type - The WSDL port type for the virtual service.
2. Virtual Service Name - The key that is given to the URL of the export for the associated proxy gateways to create a dedicated URL for the virtual service. You can change the generated value to something more relevant. For example, the default value shown in the image is accountExport1_accountHttpService, which you can change to Account or AccountService.
3. Virtual Service URLs - Network addressable endpoints that are exposed for the virtual service via the associated proxy gateways.
4. Endpoint URLs - An ordered list of network addressable endpoints that are used to forward messages for this virtual service. These are available within the Target and AlternateTarget elements of the SMO after a Gateway Endpoint Lookup mediation primitive.
5. Enable Virtual Service - must be selected if the virtual service will be available to a Gateway Endpoint Lookup mediation primitive when it queries the built-in configuration store.
6. Advanced Service Properties - A series of user defined key value pairs associated with the virtual service. These are available within the EndpointLookupContext of the SMO, after a Gateway Endpoint Lookup mediation primitive.

When you click Save, the widget returns to the original list of virtual services associated with the proxy group.

Example scenario:

A fictitious bank have web service enabled all service providers. They must put in place security and auditing across all service providers. To complete this, the bank create a proxy gateway. In the future the bank want to add more service providers without redeploying the application.

Selecting service gateway patterns

This section looks at some generic scenarios where you can use each of the service gateway patterns.

- **Web service based service gateway** - If a web service only service gateway is required, then use the proxy gateway pattern. A proxy gateway provides built-in routing capabilities and client side WSDL resolution. The other patterns, such as the dynamic service gateway, will need extensive user code to provide similar functionality.
- **Messaging or HTTP based service gateway with XML data encoding** - If the encoding style of the messaging and HTTP service gateways is XML serialization of business objects, then use the dynamic service gateway. WebSphere ESB provides an option to automatically parse incoming messages to the gateway. Therefore, access to the body of the message can be achieved using the dynamic service gateway pattern.
- **Messaging or HTTP based service gateway with non-XML data encoding** - If access to the body of the message is not required, use the dynamic service gateway. If access to the body of the message is required, use the static service gateway.
- **Protocol switching service gateway with XML data encoding** - When a multi-protocol service gateway is required and message encoding on all protocols is XML serialization of business objects, use the dynamic service gateway. WebSphere ESB provides an option to automatically parse incoming messages to the gateway. Therefore if access to the body of the message is required, use the static service gateway.
- **Protocol switching service gateway with non-XML data encoding** - If access to the body of the message is not required, use the dynamic service gateway. If access to the body of the message is required, use the static service gateway.

Aggregation

Aggregation is a powerful and important enterprise service bus (ESB) pattern. An inbound request might map into several individual outbound requests. The responses from these requests can be used to enrich a final request to a service or can be aggregated into a single response for the original request.

Common scenarios are:

- A car comparison insurance quote system where a single request is submitted to an ESB that causes multiple insurance service providers to be queried before the best quote is returned.
- A holiday booking request arrives at the ESB and needs to be forwarded to separate hotel, flight and car hire booking systems, before a combined response can be returned.
- A store ordering system, where a salesperson can submit a request for a number of orders and have them dispatched. The original message coming into the ESB identifies the customer for each order via a customer ID. So, the order request message needs to be enriched with each customer's shipping details so that the dispatch step has all the information it needs to succeed.

When creating aggregation solutions using WebSphere ESB there are several core parts of the programming model that you must understand before going into the details of the patterns:

- The Fan Out mediation primitive
- The Fan Out context
- The Fan In mediation primitive

- The Service Invoke mediation primitive
- Aggregation block
- Shared context
- Special considerations

Fan Out mediation primitive

The Fan Out mediation primitive is normally used at the start of an aggregation. It provides the ability to either branch (Once mode) the mediation flow, or fire its output terminals multiple times (Iterate mode), based on the size of a repeating structure within the Service Message Object (SMO). For example, if the inbound message contains a structure that needs to be forwarded onto two services, you can branch the flow at the Fan Out mediation primitive. To call a service multiple times, each with individual requests from a repeating structure, you can use the Iterate mode of the Fan Out mediation primitive. You can configure the two modes from the Fan Out mediation primitive properties view, as shown in Figure 203.

Figure 203. The Fan Out mediation primitive properties view

The mediation primitive is configured for Once mode by default, but this can be changed by selecting the **for each element in XPath expression** button and specifying the location of the repeating structure within the SMO.

Fan Out context

The Fan Out context is used to store the current item of a repeating element when Fan Out is used in Iterate mode. The Fan Out context contains an integer field containing the index of the occurrence of the repeating element along with an element containing the occurrence itself. For instance, if there was a repeating element of an element called *order*, the Fan Out context would look like this:

```
<FanOutContext>
 <iteration>0</iteration>
  <occurrence>
   <order>
    <id>1234567890</id>
   </order>
  </occurrence>
</FanOutContext>
```

Fan In mediation primitive

The Fan In mediation primitive is a partner to the Fan Out mediation primitive. The Fan Out mediation primitive is used to decide how many messages to generate, whereas the Fan In mediation primitive is used to decide when the

aggregation can be considered complete. The Fan In mediation primitive is always partnered with a Fan Out mediation primitive in the same flow (for example, a Fan Out/Fan In mediation primitive pairing can not expand across a request/response flow boundary), and the Fan In mediation primitive acts as a decision point for when to leave the aggregation block, explained later in this section. The Fan In mediation primitive receives a number of messages until a decision point is reached, at which point the last message that was received is propagated to the output terminal. The Fan In mediation primitive can only be used in combination with the Fan Out mediation primitive.

Three types of decision points for the Fan In mediation primitive are supported:

- **Simple count**: the output terminal is fired when a set number of messages is received at the input terminal. However, more messages might be sent out by the Fan Out mediation primitive, because the Fan In mediation primitive count decision point can be reached more than once, resulting in multiple firings of the output terminal.
- **XPath decision**: the output terminal is fired if an XPath expression evaluation of the incoming message evaluates to *true*. It is possible for the output terminal to fire multiple times.
- **Iterate**: the Fan In mediation primitive waits to receive all messages produced by the corresponding Fan Out mediation primitive. All of the occurrences in the repeating element have been processed.

Figure 204 shows these three options, which are available in the Fan In mediation primitive properties view.

Figure 204. The Fan In mediation primitive properties view

The Fan In mediation primitive has two input terminals (**in** and **stop**), two output terminals (**out** and **incomplete**) and a fail terminal (**fail**). The input terminals are wired to accept a message and the other terminals are wired to propagate a message.

The **in** terminal accepts input messages until a decision point is reached. When a decision point is reached the **out** terminal is fired. Although the **out** terminal is fired when a decision point is reached, the **in** terminal can still accept input messages, under some circumstances. For example, if the count value is reached,

the **out** terminal is fired. However, if the Fan Out mediation primitive is still sending messages, the Fan In mediation primitive can still receive messages.

You can stop a Fan In operation by wiring the preceding mediation primitive to the Fan In **stop** terminal. The **stop** terminal causes the **incomplete** output terminal to be fired, and this stops the associated Fan Out mediation primitive from sending any more messages. The **incomplete** output terminal is also fired if a timeout occurs. Table 59 on page 670 describes terminal behaviour further.

Service Invoke mediation primitive

The Service Invoke mediation primitive is used between the Fan Out and Fan In mediation primitives to call the services required. The Service Invoke mediation primitive is discussed in the "Invocation of Services" on page 631 chapter.

Aggregation block

When you create an aggregation solution, the Fan Out mediation primitive declares the start of the aggregation and a Fan In mediation primitive symbolizes the end of the aggregation. The logic between these two mediation primitives can be considered a repeating block if the Fan Out is configured in Iterate mode, or branching if configured in Once mode, as shown in Figure 205.

Figure 205. Example of an aggregation block

Shared context

WebSphere ESB has three user-defined data store contexts within the SMO:

- Data stored in the Transient context is available within either a request or response flow.
- Data stored in the Correlation context is available in both the request and response flows.
- Data stored in the Shared context is available in all branches and executions of an aggregation block.

To highlight the difference Figure 206 on page 669 shows an example.

Figure 206. Shared context example

In the case of this example, the SMOs passing through branch 1 and 2, and therefore the Transient and Correlation contexts, are different copies. This means that any changes made to the SMO in one branch are not made to the other branch. The Shared context area of the SMO has been designed for sharing aggregation data between the different branches of the flow inside an aggregation block. Access to the Shared context is only available inside the flow which contains the aggregation block that uses it.

Aggregation mediation programming model special considerations

When using aggregation capabilities consider these points:

- The Shared context is part of the SMO, and is designed for use inside aggregation blocks because the data put into it is available in all the branches of a flow. The design of the Shared context structure is essential to a successful aggregation design. The goal is to store the response from each service invocation into a structure that can then be converted into a single response message after the aggregation block.
- The Shared context is shared across SMOs and should be considered read-only outside the aggregation block.
- At development time it is difficult to determine which SMO body type will be emitted from the Fan In mediation primitive output terminal. Mediation primitives after the Fan In mediation primitive should be used to generate a new message body using the information in the Shared context.
- If a message arrives at a Fan Out mediation primitive, one or more output terminals of the Fan In mediation primitive will fire (assuming no ServiceRuntimeExceptions occur within the aggregation block, for example, as a result of running a Fail mediation primitive), even if a message is not received by the Fan In mediation primitive. Table 59 on page 670 highlights this behaviour.

  Note: In some circumstances it is possible that none of the messages sent by the Fan Out mediation primitive arrive at the corresponding Fan In mediation primitive. Using Figure 206 to illustrate, consider this example: if the second output terminal on primitives branch 1 and branch 2 are fired, this does not cause the messages to arrive at the Fan In mediation primitive, because there is no wired path. This is represented in Table 59 on page 670 by the message result “No”.

Table 59. Which output terminal is fired from the Fan In mediation primitive

Fan In mediation primitive mode	Message arrives at Fan In mediation primitive	Output terminal fired	SMO of Fan In mediation primitive
Iterate	No	Complete	Message body empty; SMO headers and context are the same as the message received at the Fan Out
Iterate	Yes (**in** terminal)	Complete	Last message received at the Fan In mediation primitive input terminal
Iterate	Yes (**stop** terminal)	Incomplete	Last message received at the Fan In mediation primitive input terminal
Simple count/XPath	No	Incomplete	Message body empty; SMO headers and context are the same as the message received at the Fan Out
Simple count/XPath	Yes (**in** terminal)	Incomplete/Complete *	Last message received at the Fan In mediation primitive input terminal
Simple count/XPath	Yes (**stop** terminal)	Incomplete	Last message received at the Fan In mediation primitive input terminal
Iterate/Simple count/XPath	Yes, timeout value expired and decision type not complete	Incomplete	Last message received at the Fan In mediation primitive input terminal
Iterate/Simple count/XPath	Yes, timeout value expired and decision type complete	Complete	Last message received at the Fan In mediation primitive input terminal

* Depends on if the condition (Simple count or XPath) evaluated to **true** for the last message that was received

Aggregation patterns

This section explains aggregation patterns and how they are constructed.

The patterns are based on the aggregation-related **Common Usage Patterns**, as discussed in chapter 2:

- **Aggregation of data from multiple sources**: This pattern is useful if the results from multiple service invocations need to be combined into a single message.
- **Batch processing with message enrichment**: This pattern is useful if the message to be processed contains a repeating element, and each element needs to be enhanced separately.
- **Batch processing requiring embedded aggregation**: This pattern is useful if the scenario is the basic batch processing pattern described in the second pattern, but the enhancement requires information from multiple sources.
- **Nested aggregation:** This pattern is useful if the scenario is the basic batch processing pattern described in the second pattern, but the repeating elements themselves contain repeating elements.

Aggregation of data from multiple sources:

This scenario is based on an insurance quote comparison website that involves a web interface that allows users to specify details of their quote.

A business processing engine controls the process and submits a single request to an aggregation mediation, which submits this request to multiple insurance firms. The responses are then aggregated into a single response for the business process to manipulate before displaying the request to the user.

Design the aggregation

The input into the flow is the InsuranceQuote interface, which specifies a Quote business object with the relevant data. The first step in planning the aggregation architecture is to determine if all data required to call the subsequent services are available in the inbound message, or if message enrichment is required. In this scenario, the Quote object contains all the information for both services. If this is not the case, then use a combination of this pattern and the batch processing with message enrichment pattern.

The mediation flow includes a Fan Out mediation primitive (operating in Once mode) receiving the Quote message as input and branching the logic to each insurance back-end service (InsuranceCompanyA and InsuranceCompanyB). Before each Service Invoke mediation primitive, an XSLT mediation primitive converts the body of the message to the correct format, and immediately after the Service Invoke mediation primitive, a Message Element Setter mediation primitive stores the response within the Shared context for later retrieval when building the aggregated response. These two branches are then collected by a Fan In mediation primitive (operating in Count mode). When the second message arrives at the Fan In mediation primitive, its output terminal fires with this message, and then a final XSLT mediation primitive transforms the values within the Shared context into the aggregated response message. This is illustrated in Figure 207.

Figure 207. Aggregation Fan Out branching

In this case, the Shared context contains the parts of the response messages that need to be stored to generate the aggregated response, as show in Figure 208.

Figure 208. Shared context response messages

This is a simple but powerful usage pattern for aggregation, especially when it is combined into a more complex scenario.

Batch processing with message enrichment:

This scenario is based on a store ordering system, where a salesperson can submit a request for a number of orders and have them dispatched. The original message that comes into the mediation flow identifies the customer for each order through a customer ID. The order request message needs to be enriched with each customer's shipping details so that the dispatch step has all the information it needs to succeed. This enrichment is done within the request flow by calling out to a customer information system, implemented using a Service Invoke mediation primitive.

Design the aggregation

The ordering and dispatch systems in this scenario are batch systems, where multiple records arrive within the same request message. However, the customer information system is implemented to receive a single record. After the ESB receives the batch message, each of the occurrences of the repeating element in the request message is processed individually, and then a batch message appropriate to the dispatch system is created. Therefore, aggregation is needed.

To explore how this basic scenario is built, you must first examine the business objects used.

The inbound message into the ordering system from the sales person includes a batched array of SalesOrder objects tagged by a batchID, as shown in Figure 209

Figure 209. Batch systems business objects

The customerID within each SalesOrder is submitted to the customer service, which provides the **Customer** business object, as in Figure 210.

Figure 210. The Customer business object

An enriched request, including the full customer details, is submitted to the BatchDispatch service with the request structure as shown in Figure 211 on page 673.

Figure 211. Batch request structure

Figure 212 then shows the request provides the following response structure that will be transformed into the sales response.

Figure 212. Batch response structure

Shared context

The structure of the Shared context is important in an aggregation design. The goal is to build an array of DispatchOrder objects so that a message can be created to send to the dispatch system. Each DispatchOrder is created from data in the original SalesOrder and is enhanced with further data from the customer information system. Next, an aggregated picture of all of these individual orders is built by moving the orders into an array of DispatchOrder objects as shown in Figure 213.

Figure 213. DispatchOrder array

The current field is used in each iteration of the Fan Out mediation primitive to create the DispatchOrder for the customer in question. This is then appended to the aggregationStore, which can be seen as a running total of the aggregation so far.

The flow detail

Now take a look at the flow in detail, starting with the aggregation mediation primitives, Fan Out and Fan In. The Fan Out mediation primitive is set to Iterate mode for the repeating elements called *SalesOrders*, in the request message. The Fan In mediation primitive setting **Fire output terminal when** is set to the option **the associated Fan Out primitive has iterated through all messages** , which ensures the mediation primitive waits until all messages from the Fan Out mediation primitive have been processed. Figure 214 shows the flow and Figure 215 shows the Fan Out and Fan In mediation primitive properties.

Figure 214. Fan Out/Fan In flow detail

Figure 215. Fan Out/Fan In properties view

A Message Element Setter mediation primitive called *setBatchID* is located after the Fan Out mediation primitive. This Message Element Setter mediation primitive is used to copy the batchID from the request message into the Shared context, as depicted in Figure 216.

Figure 216. Message Element Setter properties

Notice that in this example, the current iteration's output is built from the customer details returned from the service invocation. Additional information

(itemID and the quantity) has been copied from the current element iteration's occurrence in the Fan Out context, as shown in Figure 217.

Figure 217. Transform inline map

The complete results from the current aggregation iteration in the current object of Shared context now show. The next mediation primitive in the flow is a Message Element Setter mediation primitive called aggregateResult, which appends the current object into the aggregationStore in Shared context, as depicted in Figure 218.

Figure 218. Message Element Setter aggregateResult properties

The aggregation flow used for each element is now complete. The elements aggregationStore and batchID must be converted into the batchDispatchRequest needed for the Callout node, as shown in Figure 219.

Figure 219. SMO mapping

The key to a successful aggregation flow is to build up the required information in each iteration in a current object, and then to attach current objects into the running total aggregationStore in the Shared context.

Batch processing requiring embedded aggregation:

This scenario combines the two previously described examples, where an aggregation of data from multiple sources is embedded within a batch processing aggregation.

The scenario is a personnel-based application that creates payslip information for specific employees. The back-end service is represented by a Callout node and requires full employee records, rather than the employee IDs, which arrive at the mediation flow. These employee records are created by the mediation from two sources, an HR system and a Payroll system. The former stores information concerning the employee's name, address, and vacation entitlement, while the latter contains information regarding salary and department.

Design the aggregation

The input to the flow is an array of employee identifiers, while the output is an array of status objects, indicating whether the payslip processing succeeded or not, as shown in Figure 220

Figure 220. Flow input and output

The HR and Payroll systems that are called from within the mediation flow can only handle a single record, as shown in Figure 221.

Figure 221. HR and Payroll Record

The outer aggregation iterates through all the employeeIDs, while the inner aggregation builds a response aggregated from the HR and Payroll services. This is depicted in Figure 222 on page 677.

Figure 222. The inner and outer flow

In this case, after the two services have been called successfully, the HRRecord and PayrollRecord must be combined into a single EmployeeRecord. The EmployeeRecord must then be appended to an array of records to comply with the interface for the Payslip service, which is called via the Callout node, as in Figure 223. Consequently, the same model for Shared context as previously described is used.

Figure 223. EmployeeRecord array

After the data has been aggregated and batched, the Payslip service is called, which takes an array of EmployeeRecord objects and returns an array of PayslipStatus objects, as shown in Figure 224.

Figure 224. PayslipStatus object

Nested aggregation:

This scenario is based on a supermarket company, where each outlet sends multiple restocking orders to a central company-wide system that batches the requests into a single message. This single message is submitted to the mediation flow where, for each order and for each branch, a separate invocation to the order system should occur.

Design the aggregation

The ordering system only accepts a single record, whereas the input into the mediation module is a nested batch message. This introduces the need for aggregation, and in this case nested aggregation, because the batch request message is repeated on an outlet basis, and then repeated on an order basis. Orders are sent individually; the response from each order is stored into a single response message, which can be aggregated later. This single response message shows the status of all the elements in the batch request. To explore how this basic scenario is built, first examine the business objects used, as shown in Figure 225.

Figure 225. Business objects for batch order example

The GroupsOrders request is a batched array of Outlets tagged by an ID, and Outlets contains a batched array of Orders. Figure 226 shows the result.

Figure 226. Result of batched order example

Shared context

The structure of the Shared context is essential to a successful aggregation design, and it is even more critical within a nested aggregation. The goal is to store the response from each Order into a structure that can then be converted into a single response message. The initial request is at a company-wide scope, the first aggregation is at the Outlet scope, and the second aggregation is at the Order scope, as depicted in Figure 227.

Figure 227. Aggregation levels

Understanding the aggregation structure helps you define the correct Shared context. The logic pattern within the mediation flow is:

- Aggregate on an Outlet basis.
  - Aggregate on an Order basis.
    - Call the Order service.
    - Store the current Order service within the current Outlet response.

  - Repeat for each Order associated with the Outlet.
  - Store the current Outlet in the company-wide aggregation store.
- Repeat for each Outlet.
- Convert all results into a response message.

With this pattern you can store the overall company-wide response, the current Outlet response, and the current Order response in the Shared context, as shown in Figure 228.

Figure 228. Shared outlet responses

The flow detail

Now look at the flow in detail. It is separated into three sections:

- Company-wide scope
- Outlet or Store scope
- Order scope

This is shown in Figure 229.

Figure 229. Aggregation block

Error handling within an aggregation block

To handle an error condition within an aggregation block a Stop or Fail mediation primitive can be used. Alternatively, you can wire to the Stop input terminal of the Fan In mediation primitive.

- The Stop mediation primitive stops the current branch of execution and the aggregation will continue to either another branch or iteration.
- The Fail mediation primitive terminates the entire mediation flow and return a ServiceRuntimeException to the caller of the mediation. In general, the Fail

mediation primitive should not be used within an aggregation, unless you want to terminate the processing of the entire flow immediately.

- The Fan In mediation primitive provides a Stop input terminal that can be wired to signal that the aggregation should be terminated. When a message arrives at the Stop terminal the incomplete output terminal is fired. Additional mediation primitives can then be wired to handle this situation and if required this could include a Fail mediation primitive.

The Fan In mediation primitive includes a property that specifies the timeout value. The timeout period starts when the associated Fan Out mediation primitive fires an output terminal for the first time. If a message arrives at the Fan In mediation primitive **in** terminal after this timeout period, it is considered late, and the incomplete terminal is fired. The default value for the timeout property is `-1`, which means there is no timeout, and no messages are considered late.

Asynchronous parallel processing

To reduce the overall processing time of an aggregation you can call multiple services concurrently, instead of processing each service in series.

A branched aggregation block such as shown in Figure 230 will not, by default, be processed in parallel.

Figure 230. Branched aggregation block

To enable multiple service invocations to occur before the processing of response messages you must complete the following steps:

1. Specify the Invocation style property on each Service Invoke mediation primitive to be **Async**, as shown in Figure 231.

Figure 231. Service invoke mediation primitive property details

2. If the mediation flow component is configured to run under a Global Transaction, ensure that the component Reference that is associated with the

Service Invoke mediation primitive has a qualifier for Asynchronous invocation that is set to **Call**, as shown in Figure 232.

Figure 232. Setting the Asynchronous invocation qualifier

If a global Transaction is configured, and the Asynchronous invocation setting is **Commit**, then request messages are never sent, because the service request cannot be sent until the transaction commits.

3. An additional step of configuration is required when the aggregation block processes in Iterate mode. Using the Fan Out mediation primitive you can specify the number of concurrent unprocessed requests that can be made before the response messages are processed. By specifying an integer value in the **Check for asynchronous responses after** `X` **messages have been fired** field, as shown in Figure 233, the runtime completes the processing of `X` requests and then processes the response messages before further iterations are processed. This way you can restrict the number of parallel requests being processed by back-end services, such as the Payroll and HR services. To process all entries within the array before the handling of any response messages, select **Check for asynchronous responses after all messages have been fired**.

Figure 233. Setting the number of requests that should be processed

By using the Invocation style and Asynchronous invocation settings both Payroll and HR services are called prior to either of the response messages being processed.

Performance considerations when using aggregation

When using aggregation, there are some performance considerations to be aware of.

Aggregation patterns often require multiple requests to be built from a single input request and multiple responses to be merged into a single response. In the

examples shown in previous Aggregation-related sections this is done using BOMapper mediation primitives. If there are no complex operations required in these transformations, and the data is simply being moved from one part of the message to another, then the performance will be significantly improved by using BOMapper mediation primitives instead of XSL Transformation (XSLT) mediation primitives. This is even more relevant for larger messages. If you have complex operations, only use XSLT mediation primitives. The Data Transformation section discusses these mediation primitives in more detail.

Batch processing considerations

In the batch processing scenario it is important to consider the impact that processing large batched requests may have on other applications hosted on the same server. Processing requests with a large number of repeating elements; for example, a large number of sales orders, can use a significant amount of system memory, which will remain in use until all processing is complete. Depending on the response times of the services being called within the aggregation block processing may slow down system resources for long periods. If the same server used for processing batch requests is also being used to host applications that need to respond quickly to requests, you can see a degradation in service while the batched request is being processed. You can avoid this issue by processing large batched requests on a separate server or by processing the requests during periods of low system usage.

For the reasons explained before, it is important to minimize the amount of memory used when processing large batched request messages. When using the batched processing pattern, the body of the original request message is rarely required inside the aggregation block because the Fan Out mediation primitive stores the current instance of a repeating element in the Fan Out context. In this example it is the sales order that is being processed. At the end of the aggregation block the response data is held in the Shared context and it is used to build a new response message body. If the request is large, then significant savings in memory usage and reductions in processing time can be achieved by deleting the message body immediately after it has been processed in the Fan Out mediation primitive, by using the Message Element Setter mediation primitive, as shown in Figure 234.

Figure 234. Deleting the message body

Other memory considerations

After an aggregation block is completed the responses that have been stored in the Shared context are combined into a new response message body. This is done immediately after the Fan In mediation primitive, as shown in the BOMapper in Figure 207 on page 671 and in Figure 214 on page 674.

At this point the contents of the Shared context are not typically required and performance is improved if Shared context is not propagated through the remainder of the flow. If you do not require any of the fields in the context then leave the context unmapped. Otherwise, expand the Context element and map only the parts that are required in the rest of the mediation flow, as shown in Figure 235.

Figure 235. Aggregation context mapping

Error handling in the mediation flow component

This topic provides information on the various ways to deal with errors in the mediation flow, including ways to use the Stop and Fail mediation primitives, where to look for fail information in the message, and how to handle modelled and unmodeled faults.

Fail terminal

Mediation primitives that process messages have a fail terminal. When there is a failure in the mediation primitive, the fail terminal is called and exception information is sent, along with the input message. The exception information is stored in the failInfo element in the message context. A message showing the failInfo is shown in Figure 236 on page 684. You must wire the fail terminal of a mediation primitive to another mediation primitive in order to access the failInfo.

Figure 236. The failInfo element in the message context

Runtime failure in the mediation primitive

When a failure occurs in a mediation primitive, the unwired fail terminal sends a runtime exception, and the flow is considered to have failed. If the mediation flow component is running under a global transaction (default), primitives that use resources that participate in the global transaction will rollback.

Figure 237. Runtime failure and rollback transaction

In Figure 237, assume that a global transaction is not configured, and a failure occurs in the Custom mediation primitive. As the fail terminal has not been wired, the flow fails. The log function would still complete as the mediation flow component is not under a global transaction, and therefore still logs to the database.

When the flow is reconfigured, a global transaction is configured, and a failure occurs again in the Custom mediation primitive, the flow fails. If the Log mediation primitive has its transaction mode property set to *same*, the log entry will be rolled back as a part of the global transaction, and no entry will occur in the database. If the transaction mode property was set to *new*, then the log to the database commits outside the global transaction so an entry will still occur in the database.

Modelled faults

WSDL operations have three types of messages:

1. input
2. output
3. fault

Modelled faults are business error conditions, that are defined as a fault message in a WSDL operation. In the mediation flow component, modelled faults are handled through Input Fault and Callout Fault nodes, which have a terminal for each unique type of fault message. The Input and Callout Fault nodes are created in the mediation flow editor when there is a modelled fault defined in the source or target operation.

- An **Input Fault node** is created when a source operation has a modelled fault message defined. The Input Fault node has an input terminal for each fault message type that is defined in the source operation. Any message that is sent to an Input Fault node will result in a modelled fault error message being returned from the source operation. Wiring to this node means that the fault message is returned to the service requester. The Input Fault node is created in the request and response flows.
- A **Callout Fault node** is created in the response flow for each target operation that has a modelled fault defined. The Callout Fault node has an out terminal for each fault message type that is defined in the target operation. When a modelled fault occurs, the Callout Fault node sends a message to the mediation primitive or node to which it is wired.

Figure 238. Example of a modelled fault

As shown in Figure 238, an example of when modelled faults can be used within the mediation flow component, is if a fictitious company offers a tiered level of service. The company offers priority booking information about upcoming theatre shows to Gold customers, who pay a premium. Standard customers must manually check the website for any information. An input message can be sent to a Database Lookup mediation primitive that retrieves the membershipType and sets an isValid property in the message. The message then goes onto a Message Filter mediation primitive, to route the message based on the value of isValid. If the membershipType is valid (Gold), the message is sent to the target operation. If the membershipType is not valid, the message is returned to the service requester, through the Input Fault node.

Unmodeled faults

Errors that are returned, and are not defined as WSDL faults are called unmodeled faults. There is no Input or Callout Fault node created for these types of faults in the mediation flow editor. In this case, the input message type is sent to the Callout Response node's fail terminal. The failure information is captured in the failInfo element of the message context.

To handle an unmodeled fault, you can wire the fail terminal of a Callout Response node to a mediation primitive. For example, you could wire the fail terminal on a Callout Response node to a Message Logger mediation primitive to log all unmodeled faults. You can set a property on the Callout Response node to determine whether the entire request message or just the message header information should be passed out of the fail terminal (there can be a performance overhead if you choose to pass the entire request message).

If the fail terminal on the Callout Response node is not wired and an unmodeled fault is received, a mediation runtime exception will occur.

Stop mediation primitive

The Stop mediation primitive has one input terminal and no out terminals. When the fail or out terminal on a mediation primitive is wired to the Stop mediation primitive, that particular branch comes to an end.

The mediation flow editor generates warnings if the out terminal of a mediation primitive is not wired. The Stop mediation primitive is used to stop these warnings. In the runtime, out terminals that are not wired are automatically propagated to stop.

In Figure 239, the fail terminal of the Message Filter mediation primitive is wired to a Message Logger, so that the failed messages can be stored. To indicate that no more actions are needed, the Message Logger is wired to a Stop mediation primitive.

Figure 239. Stop mediation primitive

Fail mediation primitive

The Fail mediation primitive can be used if you want to stop the flow and send a runtime exception. You can wire the out or fail terminal on a mediation primitive, to the Fail mediation primitive. If the mediation flow component is running under a global transaction, mediation primitives participating in the global transaction will rollback.

Faults in the Service Invoke mediation primitive

There are two types of fault that can occur at the Service Invoke mediation primitive:

- Modelled faults, which are sent through the out terminal.
- Unmodeled faults, which are sent through the fail terminal.

Error flow

A mediation flow component has an error flow for each source operation. The error flow acts as a catch-all for messages that are propagated from any unwired fail terminal on any primitive or node, in either a request or a response flow.

By default, an error flow consists of:

- An Error Input node, which has a catchAll terminal, with the type set to anyType. The Error Input node propagates the Service Message Object (SMO) from the unwired terminal, that contains the error information.
- An Input Response node for (request and response) operations. You can use this node to return a message from the source operation.
- An Input Fault node, created when a source operation has a WSDL fault message defined. The Input Fault node has an input terminal for each fault message type that is defined in the source operation. Any message that is sent to an Input Fault node results in a WSDL fault error message being returned from the source operation.

You can wire mediation primitives to the Error Input node to capture error information. For example, a Message Logger mediation primitive is wired to log the SMO. You can also put your error handling logic in a subflow, so that it can be reused in multiple error flows.

You can use an error flow to audit any unhandled errors that may occur in the operation request or response flows. For example, as shown in Figure 240, you can use a Message Logger mediation primitive to capture the SMO, and then wire the Log mediation primitive to a Fail primitive.

Figure 240. An example error flow to return an unmodeled fault

An error flow can use the information available in the failInfo element of the SMO and return a modelled fault message. To do this, within the error flow, you can map the failureString from the failInfo to the modelled fault message and wire it to the Input Fault node. This is shown in Figure 241 on page 688.

Figure 241. An example error flow to return a modelled fault

Common patterns of usage for error handling

When a service component is called synchronously or asynchronously unexpected runtime errors can occur. In SCA, error handling is put in place to make sure that a message is not lost. This topic provides information on how messages are handled by the SCA runtime when an error occurs. You will find information for both synchronous and asynchronous invocation styles.

Synchronous invocation

A transaction is a unit of activity, within which multiple updates to resources can be made dependent such that all or none of the updates are made permanent. Transaction qualifiers can be used to control transactional behaviour of service invocations.

Note: In this section, we have assumed that all components are set to run in a global transaction.

When a service component is called synchronously, and performs a synchronous invocation, both the service requester and the service provider run in the same thread of execution. The calling component within WebSphere ESB is blocked until a response is received from the provider.

Figure 242. Synchronous invocation error handling

As shown in Figure 242 there are no queues in a synchronous flow. If an unhandled error occurs at any point in the flow, any active transactions will rollback and an unmodeled fault is returned to the service requester.

Asynchronous invocation

When a service component is called asynchronously, the service requester and the service provider run in different threads of execution. The SCA runtime returns control to the requester immediately.

Figure 243. Asynchronous invocation error handling

Figure 243 shows how errors can be handled within an asynchronous flow. The flow has been split into transaction boundaries (A, B, C, D), each new boundary is marked by a new queue (1, 2, 3, 4). If an error occurs within a transaction boundary, the message is rolled back to the queue at the start of the transaction. For example, if an error occurs within the mediation flow component, the message is rolled back to queue number 2, because it is within the transaction boundary B.

Figure 244. Asynchronous invocation error handling without queue 2

By default, to improve performance, when the first asynchronous invocation is made, queue 2 is omitted from the flow. As shown in Figure 244, one thread of execution will start from queue 1. If an exception occurs at the mediation flow component, the message will be sent back to queue 1, because it is within transaction boundary A. The message is redelivered to the export and the previously omitted queue 2 is reintroduced as shown in Figure 245 on page 690. Any subsequent retries, will include queue 2. This performance optimization occurs automatically. Specifically JMS, MQ and asynchronous SCA bindings all have this optimized behaviour.

Figure 245. Asynchronous invocation error handling

As a default, when a message is rolled back to an internal module destination, SCA retry logic indicates that the message will be redelivered up to five times. If the message fails again, it will be automatically forwarded to the defined exception destination. This can be viewed via the Service Integration Bus Explorer in the Administrative Console. If the Failed Event Manager is enabled, messages on the exception destination are removed and stored in a database as failed events. Each module has its own internal exception queue. Binding retries can be configured at the binding, or on the messaging provider. Binding retries occur at least once, so that the Failed Event Manager can be aware of any fault.

The Failed Event Manager is a web-based client for working with and resubmitting the failed invocations. It is an integration application and is available in the Administrative console. It displays the number of failed events and provides a number of search capabilities. Messages that exceed their retry count are automatically retrieved by the Failed Event Manager. Asynchronous export and import bindings can be configured to send failed messages to the Failed Event Manager before they exceed their retry count.

Mixed invocation styles

When a service requester calls a service provider, the invocation can be made either synchronously or asynchronously. There are situations where the service requester makes a synchronous call, but the service provider must be called asynchronously by the mediation flow component. In this occasion, queues 1 and 2, and boundaries A and B will not exist, and the flow will look similar to Figure 246. In this situation, if an error occurs before boundary C, it will be sent back to the service requester.

Figure 246. Synchronous and asynchronous invocation error handling

If the service requester performs an asynchronous invocation, and the service provider is a synchronous service, the mediation flow component will call the import synchronously. In this situation, queues 3 and 4, and boundaries C and D will not exist within Figure 247. In this situation, if an error occurs after boundary B, it will be sent back to the service provider.

Figure 247. Asynchronous and synchronous invocation error handling

Testing and Debugging

This topic provides information on tools in Integration Designer. The tools can be used to test modules and components, and to debug problems.

Unit testing

Modules are typically developed independently of other modules. Within a module, a component does not need or have visibility to another component's implementation. In Integration Designer, the integration test client is the designated tool for testing modules and components in isolation. The test client features a sophisticated user interface that enables you to easily manage and precisely control your tests.

Testing components

As you develop a component, you typically want to unit test, fix any problems, and unit test the changes again. In the integration test client, you can choose the operation that you want to test, enter values, and invoke the operation to see if the results are as you expected. You can also choose to emulate components and imports, which means that you do not actually need to run them during your testing. A manual emulation event is automatically generated in the following situations:

- You have chosen to test an entire module and the integration test client has encountered a component that has not been implemented.
- You have chosen to test a set of components or a single component in a module, which means that any other unselected components in the module are automatically emulated regardless of whether they are implemented or not.

A manual emulate event is an interactive event that requires you to enter input and output parameter values, or select an exception to throw.

Example

The component HelloWorldMediation calls an external Web service, which takes a string input and returns the concatenated string. To test the operation callHello independently of the Web service, you can use an emulate event.

1. From the module assembly diagram, call the **Test Component in Isolation**.
2. In the integration test client, select an interface and an operation that you want to test.

Figure 248. Integration Test Client properties

3. Fill in the input parameter values for the selected operation.

Figure 249. Input parameters

4. Click the **Continue** button to begin invoking the selected operation with the specified parameter values.

5. If the selected component is referenced to another component or Import, then the integration test client will pause at the point where the reference is used. It also allows you to emulate the results returned by the reference, so that you can test your component even if the dependent component is not implemented.

6. Once the values for emulation are specified, click the Continue button to continue.
7. If everything goes well, then the expected result is returned.

Cross-component trace

SOA applications tend to be large and complex, and are typically composed of other applications. The most time spent in troubleshooting an SOA application is often in narrowing the problem to a particular area. This is where cross-component trace is one of your most useful tools.

The Server Logs view displays standard server console and log records. However, if you enable cross-component tracing, the Server Logs view also displays invocation records that contain the invocation data that was passed between the components in your application. The invocation records are displayed in hierarchical format in the Server Logs view, which enables you to more easily understand the relationships that exist between the records. When you enable cross-component tracing, the Server Logs view becomes an even more powerful tool for problem determination. Typically, the process of using the Server Logs view for problem determination begins when you encounter problems when testing your application using the integration test client or another tool.

To turn on cross-component trace:

1. Bring up administrative console on a running server by right-clicking the server and from the menu select Run administrative console.
2. Log on to the console, expand Troubleshooting and click Cross-Component Trace. Select your server. For the Configuration values, select Enable with data snapshot as this collects the most data, which is the recommended setting for tracing.
3. Save the configuration.

4. Log out of the administrative console and restart your server so the changes take effect.

To produce the cross-component trace, re-run your test.

To view the cross-component trace, open the Server Logs view selecting Window > Show View > Server Logs from menu bar. In the Server Logs view, you could display the traces either from a live server console or from a trace file. To display the trace, select “Load from Server Log Directory” and “trace.log” from the log directory.

You can filter the log message to locate the invocation event by selecting Invocation Types > with only Server State and Error Contents

By expanding the invocation event, you can analyze the path or “call stack” of the test message and quickly identify the failing application.

The Server Logs view provides powerful filtering and searching capabilities to help you locate useful information from a large number of log records. Detailed information about working with server console and log records in the Server Logs view is found in the Integration Designer topic F“Using the Server Logs view for problem determination”.

Note that turning on the cross component trace has a performance impact on the server. When the tracing is completed, you can turn off the cross component trace via the administrative console.

Testing Web services

You can use the Web Services Explorer to test Web services.

To test a Web service using Web Services Explorer:

1. Select the interface that the Import of the SOA application is bound to. Select **Web Service > Test with Web Service Explorer** from the context menu.
2. Enter the end points in the WSDL binding. The end point information can be retrieved from the properties of the Import.

3. After the end point is defined, you can click the operation and fill in a test message.

4. Clicking the “Go” button will instruct the Web Services Explorer to send out the SOAP message. The response message will be displayed in the status pane.

The Web Services Explorer also supports HTTPS. The SSL certificates are managed by the JRE.

Debugging mediation flows

The integration debugger allows you to debug your mediation flow component using a graphical user interface. You can control the flow execution by using breakpoints to step over the running of mediation primitives. When the server is running in debug mode, and a breakpoint is encountered, the running stops. When a component thread pauses at a breakpoint, you can control program flow or alter the state of the running component. You can use the variables view to inspect the data or message.

To debug a mediation flow, perform the following steps:

1. Start the server in debug mode by selecting **Debug** from the server's context menu.
2. Add the projects that you want to debug to the server.
3. Open the mediation flow that you want to debug.
4. Select a mediation primitive that you want the flow execution to pause at, right-click and select **Debug -> Add Breakpoint**.
5. If you want to debug within a subflow, you must open the subflow and set a breakpoint on a mediation primitive in the subflow.
6. Open the assembly diagram and launch the integration test client to test the mediation flow component.
7. Enter the test data in value editor of the integration test client.
8. Click on the Continue button to start executing the test.
9. The flow execution will pause at the first breakpoint. You can inspect the message coming in to the mediation primitive in the Variables view.
10. When you have inspected the message, you can then step over the primitive or continue the flow execution until it hits the next breakpoint.

Service Component Architecture programming

Service Component Architecture (SCA) provides a simple, yet powerful programming model for constructing applications based on service-oriented architecture (SOA).

Service Component Definition Language

Service Component Definition Language (SCDL) is an XML-based language used to describe Service Component Architecture (SCA) elements such as modules, components, references, imports, and exports.

The various artifact types that exist in SCA were designed to support some of the basic requirements of this service-oriented architecture. To start with, SCA needs a mechanism for defining a basic service component. Once there is a mechanism for defining service components, it is important to be able to make these services available to clients both inside or outside of the current SCA module. In addition to this, a construct designed to import and reference services external to the current SCA module must exist. Finally, SCA provides constructs for composing services and modules into larger applications.

SCDL definitions are organized across several files. For example, in a credit approval application, we can store the SCDL for the interface and implementation in a file called `CreditApproval.component`. References can be included in the `CreditApproval.component` file (inline) or in a separate `sca.references` file located in the module root. Any stand-alone reference is placed in the `sca.references` file. Stand-alone references can be used by non-SCA artifacts (JSP) within the same SCA module to invoke the SCA component.

Table 60. Primary artifacts that make up an SCA service module

Artifact	SCDL Definition
Module Definition	• Contained in the `sca.module` file at the root SCA project JAR
Service Components	• A module can contain 0..n service definitions • Each component definition is contained in a `<SERVICE_NAME>.component` file
Imports	• A module can contain 0..n import definitions • Each import definition is contained in a `<IMPORT_NAME>.import` file
Exports	• A module can contain 0..n export definitions • Each export definition is contained in a `<EXPORT_NAME>.export` file
References	• Two types of references – Inline (contained within a service component definition) – Stand-alone • Each component definition is contained in a `<SERVICE_NAME>.reference` file
Other Artifacts	• Other artifacts include: Java Classes, WSDL files, Other Artifacts XSD files, BPEL.

When building an SCA application, IBM Integration Designer takes care of generating the appropriate SCDL definitions. However, a basic familiarity with SCDL can help you to understand the overall architecture and assist when debugging applications.

Module definition

Service Component Architecture (SCA) defines a standard deployment model for packaging components into a service module. The `sca.module` file contains the definition of the module.

An SCA module is not just another type of package. In WebSphere ESB, an SCA service module is equivalent to a Java EE EAR file and several other Java EE sub-modules. Java EE elements, such as a WAR file, can be packaged along with the SCA module. Non-SCA artifacts (JSPs and others) can also be packaged together with an SCA service module, enabling them to invoke SCA services through the SCA client programming model using a special type of reference called a stand-alone reference.

Here is an example of a `sca.module` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:module xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
name="CreditApproval"/>
```

The diagram shows a module in IBM Integration Designer along with its associated SCDL module definition that you can view in an editor. In this example, the module type is a mediation module.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:module xmlns:mt="http://www.ibm.com/xmlns/prod/websphere/scdl/moduletype/7.0.0"
 Xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"name="StockQuote">
 <mt:moduleType type="com.ibm.ws.sca.scdl.moduletype.mediation" version="1.0.0"/>
</Scdl:module>
```

Figure 250. Showing the relationship between a module in IBM Integration Designer and the module definition

Component definition

The service component definition is included in a file called `<SERVICE_NAME>.component`. SCA components with their associated dependencies can be defined and packaged together into deployable units.

This figure provides a more detailed look at the service component definition. Each service component must have a unique name within the SCA module and it must match the file path relative to the module root. As noted on the previous slide, the service component definition is included in a file called `<SERVICE_NAME>.component`.

Figure 251. Service component definition including component name, implementation, interfaces, and references

Each service component must have a unique name within the SCA module and it must match the file path relative to the module root. Next, each service component can have one or more interfaces associated with it, which can be either Java or WSDL port type interface definitions. The interfaces associated with a service component can support either a synchronous or asynchronous interaction style with clients calling the service.

Each service component can be implemented in various ways, specified by the implementation definition. Finally, service components can invoke other service components or imports defined in the current service module. In this case, the appropriate reference must be defined to indicate which service is used. Often this type of reference is in lined in the service component definition, although it may alternatively be placed in the stand-alone references file. Each service component definition can have zero or more references to other services called by the service component being defined.

Here is an example that shows the definition for the StockQuote_MediationFlow component. Notice that the component includes definitions for a WSDL interface, two references, and an implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        xmlns:ns1="http://Resources/StockQuoteService"
        xmlns:ns2="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:ns3="http://stockquote.samp.sibx.websphere.ibm.com/RealtimeService/"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        DisplayName="StockQuote_MediationFlow" name="StockQuote_MediationFlow">
 <interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:StockQuoteService"/>
 </interfaces>
 <references>
  <reference name="DelayedServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:DelayedServicePortType">
    <method name="getQuote"/>
   </interface>
   <wire target="DelayedService"/>
  </reference>
  <reference name="RealtimeServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns3:RealtimeServicePortType">
    <Method name="getQuote"/>
   </interface>
   <wire target="RealtimeService"/>
  </reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="StockQuote_MediationFlow.mfc"/>
</Scdl:component>
```

Figure 252. Example of a component definition in Service Component Definition Language

Import definition

The import definition is included in a file called `<IMPORT_NAME>.import`. SCA imports allow clients in an SCA module to access services that are outside the current SCA module.

Like service components, imports have a name and a set of 1..N interfaces with which they are associated. Imports also have a binding attribute, which is used to describe how the external service is bound to the current module. The common binding types are indicated in the diagram.

Imports can be thought of as a special type of service component in an SCA module. Imports are valid targets in a wire definition for a service reference. This means that to a client invoking a target service the client programming model is the same whether the reference points to an import or another service component.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ns1="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        xmlns:webservice="http://www.ibm.com/xmlns/prod/websphere/scdl/webservice/7.0.0"
        xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        displayName="DelayedService" name="DelayedService">
  <interfaces>
    <interface xsi:type="wsdl:WSDLPortType" portType="ns1:DelayedServicePortType">
      <method name="getQuote"/>
    </interface>
  </interfaces>
  <esbBinding xsi:type="webservice:WebServiceImportBinding"
        endpoint="http://localhost:9080/DelayedService/services/DelayedServiceSOAP"
        port="ns1:DelayedServiceSOAP" service="ns1:DelayedService"/>
</scdl:import>
```

Figure 253. Example of an import definition in Service Component Definition Language

Export definition

The export definition is included in a file called `<EXPORT_NAME>.export`. SCA exports provide access to service components defined in an SCA module for use by clients outside of the current SCA module.

Export components include a name and a target attribute, which names the service component that is to be exported. Like import components, exports have a binding attribute that indicates how the service is bound externally. The common binding types are indicated in the diagram.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:export xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:_="http://Resources/StockQuoteService/Binding"
xmlns:ns1="http://Resources/StockQuoteService"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        xmlns:webservice="http://www.ibm.com/xmlns/prod/websphere/scdl/webservice/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        displayName="StockQuoteService" name="StockQuoteService" target="StockQuote_MediationFlow">
 <interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:StockQuoteService">
   <method name="getQuote"/>
  </interface>
 </interfaces>
```

Figure 254. Example of an export definition in Service Component Definition Language

Reference definition

SCA and non-SCA clients calling a service component need a reference to that service in order to invoke it. References can be defined either as stand-alone in the `sca.reference` file or as inline within a service composition definition.

Each reference has a name, used to look up the appropriate service by a client using the client programming model. In addition to the name, a reference also includes an interface element. The multiplicity for a reference indicates how many wire definitions can name this reference as the source. Finally, the wire definition specifies the name of the target service component or import that resolves the reference.

There are two ways to define references. The first way is to inline the reference in the service component definition. Using this approach, the references are only available to the service component in which the references are included. Another approach is to include reference definitions within the stand-alone references file. For this approach, the references can be used by a non-SCA client or by another component within the module. An example of a non-SCA component that uses a reference in the stand-alone references file is a user interface component such as a JSP that needs the ability to invoke a particular service. In order to invoke a service component, the client needs a reference so that it can use the SCA runtime to look up the appropriate service to invoke.

Figure 255. Clients can call a service component using either stand-alone or inline references

Here is an example that shows the definition for the component that includes two inline references: DelayedServicePortTypePartner and RealtimeServicePortTypePartner. Notice that the component includes definitions for a WSDL interface, two references, and an implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        Xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        Xmlns:ns1="http://Resources/StockQuoteService"
        xmlns:ns2="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:ns3="http://stockquote.samp.sibx.websphere.ibm.com/RealtimeService/"
        Xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        DisplayName="StockQuote_MediationFlow" name="StockQuote_MediationFlow">
 <Interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:StockQuoteService"/>
 </Interfaces>
 <references>
  <reference name="DelayedServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:DelayedServicePortType">
    <method name="getQuote"/>
   </interface>
   <wire target="DelayedService"/>
  </reference>
  <reference name="RealtimeServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns3:RealtimeServicePortType">
    <Method name="getQuote"/>
   </interface>
   <wire target="RealtimeService"/>
  </reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="StockQuote_MediationFlow.mfc"/>
</Scdl:component>
```

Figure 256. Example of inline reference definitions

SCA programming model fundamentals

The concept of a software *component* forms the basis of the Service Component Architecture (SCA) programming model. A component is a unit that implements some logic and makes it available to other components through an interface. A component may also require the services made available by other components. In that case, the component exposes a *reference* to these services.

In SCA, every component must expose at least one interface. The assembly diagram shown in Figure 257 on page 707 has three components. Each component has an interface that is represented by the letter I. A component can also refer to other components. References are represented by the letter R. References and interfaces are then linked in an assembly diagram. Essentially, the integration developer “resolves” the references by connecting them with the interfaces of the components that implement the required logic.

Figure 257. Assembly diagram

Invoking SCA Components

To provide access to the services to be invoked, the SCA programming model includes a *ServiceManager* class, which enables developers to look up available services by name. Here is a typical Java code fragment illustrating service lookup. The ServiceManager is used to obtain a reference to the BOFactory service, which is a system-provided service:

```
//Get service manager singleton
ServiceManager smgr = new ServiceManager();
//Access BOFactory service
BOFactory bof =(BOFactory)
        smgr.locateService("com/ibm/websphere/bo/BOFactory");
```

Note: The package for ServiceManager is `com.ibm.websphere.sca`.

Developers can use a similar mechanism to obtain references to their own services by specifying the name of the service referenced in the *locateService* method. After you have obtained a reference to a service using the ServiceManager class, you can invoke any of the available operations on that service in a way that is independent of the invocation protocol and the type of implementation.

SCA components can be called using three different invocation styles:

- **Synchronous invocation**: When using this invocation style, the caller waits synchronously for the response to be returned. This style is the classic invocation mechanism.
- **Asynchronous invocation**: This mechanism allows the caller to invoke a service without waiting for the response to be produced right away. Instead of getting the response, the caller gets a "ticket," which can be used later to retrieve the response. The caller retrieves the response by calling a special operation that must be provided by the callee for this purpose.
- **Asynchronous invocation with callback**: This invocation style is like the preceding one, but it delegates the responsibility of returning the response to the callee. The caller needs to expose a special operation (the callback operation) that the callee can invoke when the response is ready.
- **Asynchronous invocation with deferred response**: In this invocation style, the client invokes a service and then continues processing until some later time when the client makes a request to capture the response.

Imports

Sometimes, business logic is provided by components or functions that are available on external systems, such as legacy applications, or other external implementations. In those cases, the integration developer cannot resolve the reference by connecting a reference to a component containing the implementation he or she needs to connect the reference to a component that "points to" the external implementation. Such a component is called an *import*. When you define an import, you need to specify how the external service can be accessed in terms of location and the invocation protocol.

Exports

Similarly, if your component has to be accessed by external applications, which is often the case, you must make it accessible. You make it accessible by using a special component that exposes your logic to the "outside world." Such a component is called an *export*. Exports can also be invoked synchronously or asynchronously.

Stand-alone references

In WebSphere ESB, an SCA service module is packaged as a Java EE EAR file that contains several other Java EE submodules. Java EE elements, such as a WAR file, can be packaged along with the SCA module. Non-SCA artifacts such as JSPs can also be packaged together with an SCA service module. This packaging lets them invoke SCA services through the SCA client programming model using a special type of component called a stand-alone reference.

The SCA programming model is strongly declarative. Integration developers can configure aspects such as transactional behavior of invocations, propagation of security credentials, whether an invocation should be synchronous or asynchronous in a declarative way, directly in the assembly diagram. The SCA runtime, not the developers, is responsible for taking care of implementing the behavior specified in these modifiers. The declarative flexibility of SCA is one of the most powerful features of this programming model. Developers can concentrate on implementing business logic, rather than focusing on addressing technical aspects, such as being able to accommodate asynchronous invocation mechanisms. All these aspects are automatically taken care of by the SCA runtime.

Qualifiers

The qualifiers govern the interaction between a service client and a target service. Qualifiers can be specified on service component references, interfaces, and implementations and are typically external to an implementation.

The different categories of qualifiers include the following:

- Transaction, which specifies the way transactional contexts are handled in an SCA invocation
- Activity session, which specifies how Activity Session contexts are propagated
- Security, which specifies the permissions
- Asynchronous reliability provides rules for asynchronous message delivery

SCA allows these quality of service (QoS) qualifiers to be applied to components declaratively (without requiring programming or a change to the services implementation code). You can add service qualifiers using WebSphere Integration

Developer. Typically, you apply QoS qualifiers when you are ready to consider solution deployment.

Client programming model

The SCA client programming model is designed to locate a service, to create data objects, to invoke a service, and to handle exceptions that are raised by the invoked component.

The SCA client programming model provides two primary functions for clients. The programming model exposes an interface that allows clients to locate services within the current module, and once a service is located the client programming model provides a way for the client to invoke operations on that service.

Clients locate services by using the ServiceManager class. There are a few ways to instantiate the ServiceManager class, depending on the wanted lookup scope for the service.

The key interface that clients should be aware of for locating services is `com.ibm.websphere.sca.ServiceManager`. This interface includes a locateService method that returns a reference to the service implementation for the service requested. The string parameter that is passed into the locateService method represents the reference name for the service that the client wants to locate. The Java documentation for the SCA programming model is included in the Business Process Manager information center, and is also included if you choose to install the Java documentation as part of the Business Process Manager installation.

Once a client has located the appropriate service, there are two types of invocation models that can be used to make a call to an operation or method offered by the service. First, there is a *dynamic invocation* style of interaction. The key interface for this style of interaction is `com.ibm.websphere.sca.Service`. The invoke() method on this interface takes the name of the operation that you are going to invoke, along with the parameters needed to call that operation.

```
public Interface MyService {
public String someMethod(String input);

Service myService = (Service) serviceManager.locateService("myService");
DataObject input = ...
DataObject result = (DataObject) myService.invoke("someMethod", input);
```

Clients can also use a *static (type-safe) invocation* method to call a particular operation associated with a service. This type of invocation only works for interface definitions that are specified as Java. In this situation, the client casts the return from the locateService() call to the appropriate interface and can proceed calling the appropriate type safe method calls on that interface.

```
public Interface MyService {
public String someMethod(String input);

MyService myService = (MyService) serviceManager.locateService("myService");
String input = ...
String result = myService.someMethod(input);
```

Interfaces

A service component has one or more interfaces with which it is associated. The interfaces associated with a service component advertise the business operations associated with this service.

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.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        xmlns:ns1="http://HelloWorldLibrary/HelloWorld" xmlns:ns2="http://HelloService/HelloService"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        displayName="HelloWorldMediation" name="HelloWorldMediation">
 <Interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:HelloWorld">
   <scdl:interfaceQualifier xsi:type="scdl:JoinTransaction" value="true"/>
  </Interface>
 </interfaces>
 <references>
  <reference name="HelloServicePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:HelloService"/>
   <scdl:referenceQualifier xsi:type="scdl:SuspendTransaction" value="false"/>
   <scdl:referenceQualifier xsi:type="scdl:Reliability"/>
   <scdl:referenceQualifier xsi:type="scdl:DeliverAsyncAt" value="commit"/>
   <wire target="HelloServiceImport"/>
  </Reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="HelloWorldMediation.mfc">
  <scdl:implementationQualifier xsi:type="scdl:Transaction" value="global"/>7
```

Figure 258. Example of an interface definition within a SCDL component definition

These interfaces can be specified as either Java interfaces or WSDL port type interfaces. However, you can not mix Java and WSDL port type interfaces on the same service component definition. The arguments and return types for these interfaces are specified as simple Java types, Java classes, Service DataObjects, or XML Schema (for WSDL port type interfaces).

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.

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.

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.

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.

Developing service modules

A service component must be contained in a service module. Developing service modules to contain service components is key to providing services to other modules.

Before you begin

This task assumes that an analysis of requirements shows that implementing a service component for use by other modules is beneficial.

About this task

After analyzing your requirements, you might decide that providing and using service components is an efficient way to process information. If you determine that reusable service components would benefit your environment, create a service module to contain the service components.

Procedure

1. Identify service components other modules can use.

   After you have identified the service components, continue with “Developing service components”.
2. Identify service components in an application that could use service components in other service modules.

   After you have identified the service components and their target components, continue with “ Invoking components” or “Dynamically invoking components”
3. Connect the client components with the target components through wires.

Overview of developing modules:

A module is a basic deployment unit for WebSphere ESB applications. A module can contain components, libraries, and staging modules used by the application.

Developing modules involves ensuring that the components, staging modules, and libraries (collections of artifacts referenced by the module) required by the application are available on the production server.

Integration Designer is the main tool for developing modules for deployment to WebSphere ESB. Although you can develop modules in other environments, it is best to use Integration Designer.

WebSphere ESB supports mediation modules. Mediation modules are types of Service Component Architecture (SCA) module. A mediation module allows communication between applications by transforming the service invocation to a format understood by the target, passing the request to the target and returning the result to the originator.

The following sections address how to implement and update modules for WebSphere ESB.

Components

SCA modules contain components, which are the basic building blocks to encapsulate reusable business logic. Components provide and consume services and are associated with interfaces, references, and implementations. The interface defines a contract between a service component and a calling component.

With WebSphere ESB, a module can either export a service component for use by other modules or import a service component for use. To invoke a service component, a calling module references the interface to the service component. The references to the interfaces are resolved by configuring the references from the calling module to their respective interfaces.

To develop a module you must do the following activities:

1. Define or identify interfaces for the components in the module.
2. Define or manipulate business objects used by components.
3. Define or modify components through their interfaces.

   Note: A component is defined through its interface.
4. Optional: Export or import service components.
5. Create an enterprise archive (EAR) file to deploy to the run time. You create the file using either the export EAR feature in Integration Designer or the **`serviceDeploy`** command.

Development types

WebSphere ESB provides a component programming model to facilitate a service-oriented programming paradigm. To use this model, a provider exports interfaces of a service component so that a consumer can import those interfaces and use the service component as if it were local. A developer uses either strongly-typed interfaces or dynamically-typed interfaces to implement or invoke the service component. The interfaces and their methods are described in the References section within this information center.

After installing service modules to your servers, you can use the administrative console to change the target component for a reference from an application. The new target must accept the same business object type and perform the same operation that the reference from the application is requesting.

Service component development considerations

When developing a service component, ask yourself the following questions:

- Will this service component be exported and used by another module?

  If so, make sure the interface you define for the component can be used by another module.
- Will the service component take a relatively long time to run?

  If so, consider implementing an asynchronous interface to the service component.
- Is it beneficial to decentralize the service component?

  If so, consider having a copy of the service component in a service module that is deployed on a cluster of servers to benefit from parallel processing.
- Does your application require a mixture of 1-phase and 2-phase commit resources?

If so, make sure you enable last participant support for the application.

Note: If you create your application using Integration Designer or create the installable EAR file using the **serviceDeploy** command, these tools automatically enable the support for the application. See the topic, "Using one-phase and two-phase commit resources in the same transaction" in the WebSphere Application Server Network Deployment information center.

Developing service components:

Develop service components to provide reusable logic to multiple applications within your server.

Before you begin

This task assumes that you have already developed and identified processing that is useful for multiple modules.

About this task

Multiple modules can use a service component. Exporting a service component makes it available to other modules that refer to the service component through an interface. This task describes how to build the service component so that other modules can use it.

Note: A single service component can contain multiple interfaces.

Procedure

1. Define the data object to move data between the caller and the service component.

   The data object and its type is part of the interface between the callers and the service component.
2. Define an interface that the callers will use to reference the service component.

   This interface definition names the service component and lists any methods available within the service component.
3. Generate the class that implements calling the service.
4. Develop the implementation of the generated class.
5. Save the component interfaces and implementations in files with a .java extension.
6. Package the service module and necessary resources in a JAR file.

   See "Deploying a module to a production server" in this information center for a description of steps 6 through 8.
7. Run the **serviceDeploy** command to create an installable EAR file containing the application.
8. Install the application on the server node.
9. Optional: Configure the wires between the callers and the corresponding service component, if calling a service component in another service module.

   The "Administering" section of this information center describes configuring the wires.

Examples of developing components

This example shows a synchronous service component that implements a single method, CustomerInfo. The first section defines the interface to the service component that implements a method called getCustomerInfo.

```
public interface CustomerInfo {
 public Customer getCustomerInfo(String customerID);
}
```

The following block of code implements the service component.

```
public class CustomerInfoImpl implements CustomerInfo {
 public Customer getCustomerInfo(String customerID) {
  Customer cust = new Customer();

  cust.setCustNo(customerID);
  cust.setFirstName("Victor");
  cust.setLastName("Hugo");
  cust.setSymbol("IBM");
  cust.setNumShares(100);
  cust.setPostalCode(10589);
  cust.setErrorMsg("");

  return cust;
 }
}
```

The following section is the implementation of the class associated with StockQuote.

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

What to do next

Invoke the service.

Invoking components:

Components with modules can use components on any node of cluster on WebSphere ESB.

Before you begin

Before invoking a component, make sure that the module containing the component is installed on WebSphere ESB.

About this task

Components can use any service component available within a cluster on WebSphere ESB by using the name of the component and passing the data type the component expects. Invoking a component in this environment involves locating and then creating the reference to the required component.

Tip: A component in a module can invoke a component within the same module, known as an intra-module invocation. Implement external calls (inter-module invocations) by exporting the interface in the providing component and importing the interface in the calling component.

Important: When invoking a component that resides on a different server than the one on which the calling module is running, you must perform additional configurations to the servers. The configurations required depend on whether the component is called asynchronously or synchronously. How to configure the application servers in this case is described in related tasks.

Procedure

1. Determine the components required by the calling module.

   Note the name of the interface within a component and the data type that interface requires.
2. Define a data object.

   Although the input or return can be a Java class, a service data object is optimal.
3. Locate the component.
   a. Use the **ServiceManager** class to obtain the references available to the calling module.
   b. Use the locateService() method to find the component.

      Depending on the component, the interface can either be a Web Service Descriptor Language (WSDL) port type or a Java interface.
4. Invoke the component synchronously.

   You can either invoke the component through a Java interface or use the invoke() method to dynamically invoke the component.
5. Process the return.

   The component might generate an exception, so the client has to be able to process that possibility.

Example of invoking a component

The following example creates a **ServiceManager** class.

```
ServiceManager serviceManager = new ServiceManager();
```

The following example uses the **ServiceManager** class to obtain a list of components from a file that contains the component references.

```
InputStream myReferences = new FileInputStream("MyReferences.references");
ServiceManager serviceManager = new ServiceManager(myReferences);
```

The following code locates a component that implements the StockQuote Java interface.

```
StockQuote stockQuote = (StockQuote)serviceManager.locateService("stockQuote");
```

The following code locates a component that implements either a Java or WSDL port type interface. The calling module uses the **Service** interface to interact with the component.

Tip: If the component implements a Java interface, the component can be invoked through either the interface or the invoke() method.

```
Service stockQuote = (Service)serviceManager.locateService("stockQuote");
```

The following example shows MyValue, code that calls another component.

```
public class MyValueImpl implements MyValue {

 public float myValue throws MyValueException {

  ServiceManager serviceManager = new ServiceManager();

     // variables
         Customer customer = null;
         float quote = 0;
         float value = 0;

     // invoke
         CustomerInfo cInfo =
      (CustomerInfo)serviceManager.locateService("customerInfo");
         customer = cInfo.getCustomerInfo(customerID);

     if (customer.getErrorMsg().equals("")) {

         // invoke
       StockQuote sQuote =
      (StockQuote)serviceManager.locateService("stockQuote");
       Ticket ticket =  sQuote.getQuote(customer.getSymbol());
    // ... do something else ...
       quote =  sQuote.getQuoteResponse(ticket, Service.WAIT);

         // assign
          value = quote * customer.getNumShares();
     } else {

         // throw
         throw new MyValueException(customer.getErrorMsg());
     }
     // reply
         return value;
 }
}
```

What to do next

Configure the wires between the calling module references and the component interfaces.

Dynamically invoking a component:

When an module invokes a component that has a Web Service Descriptor Language (WSDL) port type interface, the module must invoke the component dynamically using the invoke() method.

Before you begin

This task assumes that a calling component is invoking a component dynamically.

About this task

With a WSDL port type interface, a calling component must use the invoke() method to invoke the component. A calling module can also invoke a component that has a Java interface this way.

Procedure

1. Determine the module that contains the component required.
2. Determine the array required by the component.

   The input array can be one of three types:
   - Primitive uppercase Java types or arrays of this type
   - Ordinary Java classes or arrays of the classes
   - Service Data Objects (SDOs)
3. Define an array to contain the response from the component.

   The response array can be of the same types as the input array.
4. Use the invoke() method to invoke the required component and pass the array object to the component.
5. Process the result.

Examples of dynamically invoking a component

In the following example, a module uses the invoke() method to call a component that uses primitive uppercase Java data types.

```
Service service = (Service)serviceManager.locateService("multiParamInf");

  Reference reference = service.getReference();

  OperationType methodMultiType =
    reference.getOperationType("methodWithMultiParameter");

  Type t = methodMultiType.getInputType();

  BOFactory boFactory = (BOFactory)serviceManager.locateService
    ("com/ibm/websphere/bo/BOFactory");

  DataObject paramObject = boFactory.createbyType(t);

  paramObject.set(0,"input1")
  paramObject.set(1,"input2")
  paramObject.set(2,"input3")

  service.invoke("methodMultiParamater",paramObject);
```

The following example uses the invoke method with a WSDL port type interface as the target.

```
Service serviceOne = (Service)serviceManager.locateService("multiParamInfWSDL");

 DataObject dob = factory.create("http://MultiCallWSServerOne/bos", "SameBO");
   dob.setString("attribute1", stringArg);

 DataObject wrapBo = factory.createByElement
  ("http://MultiCallWSServerOne/wsdl/ServerOneInf", "methodOne");
   wrapBo.set("input1", dob); //wrapBo encapsulates all the parameters of methodOne
   wrapBo.set("input2", "XXXX");
   wrapBo.set("input3", "yyyy");

 DataObject resBo= (DataObject)serviceOne.invoke("methodOne", wrapBo);
```

Invocation styles

With SCA, you can invoke service components using synchronous and asynchronous programming styles. You can assemble modules into overall solutions where asynchronous channels between service components and modules can increase the overall throughput and flexibility of the system.

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; 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.

When authoring applications in IBM Integration Designer, it is a best to explicitly set the invocation style that each of your components use to call each other. At a minimum, you want to know what invocation styles are used throughout your application while you do performance analysis or develop your error handling strategy. You certainly need to understand the interactions in your application when you consider/set your transaction boundaries. Users are often surprised to find that setting or determining invocation styles between components is not as easy a task as it seems. This section explains how to set or determine which invocation style is used at runtime, based on specific characteristics of your application.

The invocation styles that SCA provides are:

- Synchronous
- Asynchronous using one-way operation
- Asynchronous with callback
- Asynchronous with deferred response

The SCA API that is used in the Java implementation to make the invocation determines what the invocation style is at runtime:

- `invoke()`: Synchronous
- `invokeAsync()`: Asynchronous
- `invokeAsyncWithCallback()`: Asynchronous

In general, when considering an interaction from one component (source or client) to another (target), the service client determines what type of invocation is used. For example, if your source component is a Java™ component, the invocation style from source to target is determined by the particular SCA invocation API you use in the implementation, such as invoke(), invokeAsync(), or invokeAsyncWithCallback(). Each of the other components provided in Integration Designer has a set of rules that it uses to determine whether an invocation is synchronous or asynchronous.

Some components/imports are considered asynchronous:

- Long running BPEL

- Human tasks
- MQ/MQ imports
- JMS/Generic imports
- JMS/JMS imports

All invocations to components or imports of these types must be asynchronous invocations. If the calling component (or source) initiates the interaction synchronously, SCA switches the interaction to be asynchronous.

Synchronous invocation:

Service component interfaces (SCA) are always defined in the synchronous form. For each synchronous interface, one or more asynchronous interfaces can be generated.

When a service component is invoked synchronously, both the client (consumer) and the service provider execute in the same thread. The calling component within WebSphere ESB is blocked until a response is received from the provider.

Figure 259. Synchronous invocation

Asynchronous invocation:

WebSphere ESB delivers a powerful programming model for developing asynchronous applications. With asynchronous invocation in SCA, there are three types of asynchronous interaction styles available: one way, deferred response, and request with callback. With all three types of asynchronous invocation, the client receives control back immediately from the SCA runtime upon an `invokeAsync()` call.

In addition to its published APIs and tools to develop asynchronous programs using Java, WebSphere ESB also comes with a number of built-in asynchronous messaging bindings and built-in asynchronous components.

There are three different ways that the client can capture the response at a later time. First, the client can choose to discard the response entirely or if it is a call to a void method. In this case, the asynchronous invocation is said to be *one way*. Another option is for the client to call `invokeAsync()` and then continue processing until some later time when the client makes a request to capture the response. This scenario is termed *deferred response*. Finally, the client also has the option of doing an asynchronous *request with callback*. To do this, the client must first implement the ServiceCallback interface. Then, after calling `invokeAsync()`, the SCA runtime provides a callback to the ServiceCallback handler to provide the response to the client.

Figure 260. Asynchronous invocation models

SCA interfaces are always defined in the synchronous form. For each synchronous interface, one or more asynchronous interfaces can be generated. When a callback mechanism is chosen by the client, the client component needs to implement a class: `<interface name>.Callback.java`. The interface of this class is derived from the interface of the actual component that the client wants to use.

Synchronous-over-asynchronous invocation:

It is considered a best practice to avoid using synchronous-over-asynchronous invocation, which occurs when a component synchronously calls another component that has an asynchronous implementation.

When synchronous-over-asynchronous invocation is detected at run time, the Service Component Architecture (SCA) automatically changes the synchronous call to an asynchronous call. This results in a new transaction boundary and can cause one or more of the following problems:

- Lack of transaction propagation
- Multiple retries that result in multiple invocations of the same service
- Time outs when the default wait time for invocations is exceeded
- Blocked threads and thread pool depletion, which are caused when a thread is allocated for each asynchronous call and waits for a response

Most synchronous-over-asynchronous invocations occur when you have a component that synchronously calls one of the following components that typically have asynchronous implementations:

- Long running BPEL and BPMN business processes
- Human tasks
- MQ/MQ imports
- JMS/generic imports
- JMS/JMS imports
- Java POJO components that implement the ServiceAsynchImpl interface

For example, a Web Service export will always perform a synchronous call. If an export synchronously calls a long-running BPEL process, it will result in a synchronous-over-asynchronous invocation. Similarly, a microflow always performs a synchronous call for a request-response operation. If a microflow invokes an MQ import for a request-response operation, the result will be a synchronous-over-asynchronous invocation.

In general, you should try to avoid the following scenarios that may result in a synchronous-over-asynchronous invocation:

- A Web Services export calling a long-running BPEL or BPMN business process
- A Web Services export calling a message-binding import, such as an MQ or JMS import
- A BPEL microflow calling a long-running BPEL or BPMN process for a request-response operation
- A BPEL microflow calling a message-binding import (such as an MQ or JMS import) for a request-response operation

When a synchronous-over-asynchronous invocation is detected, the following warning message is displayed in the SystemOut.log file:

```
000000ac Core         W   CWSCA2011W: Service Component Architecture (SCA) is switching a synchronous call to an asynchronous call.
Request will be sent in a new local transaction. Default timeout is 115000 milliseconds.
```

Depending on your specific application scenario, there are several approaches that you can use to avoid a synchronous-over-asynchronous invocation. Many of these approaches are described in the article Process implementation types: Patterns-based design for process-based solutions. For example, if you have a microflow that invokes a long-running BPEL process over MQ for a request-response operation, you may want to incorporate a briefly-persisted process to avoid a synchronous-over-asynchronous invocation. The use of briefly-persisted processes is described in the “Briefly persisted process” section of the article.

For general information about invocation styles, see the Invocation styles topic.

SCA interactions:

SCA supports synchronous and asynchronous invocation of modules. Developers have the option of selecting the appropriate interfaces and invocation methods for their SCA interactions.

The diagram summarizes the different interface types, the supported invocation methods and models, and how data is passed between client and service.

For synchronous invocation, data is passed by reference within the same SCA module, while for asynchronous calls the data is passed by value. The table also summarizes when it is possible to use either type safe or dynamic invocation based upon the interface type. The dynamic invocation methods are always available for either WSDL port type or Java interfaces. However, in order to have type safe invocation methods available to the client a Java interface type must be used for the interface definition on the appropriate client reference.

Interface Type	Invocation Model				Invocation Methods	
	Synchronous	One Way	Deferred Response	Request with Callback	Dynamic	Type Safe
WSDL Port Type	●	■	■	■	YES	NO
Java Interface	●	■	■	■	YES	YES

● Data passed by reference in the same SCA module

■ Data passed by value

Figure 261. Summary of invocation models along with supported methods for passing data

Dynamic client invocation

There are a number of key methods and interfaces needed to support both synchronous and asynchronous interaction when using dynamic client invocation.

Table 61. Summary of key methods and interfaces for dynamic client invocation

Interface	Methods	Description
Service	Object invoke(Service String, Object)	Used to invoke synchronous service requests
	Ticket invokeAsync(String, Object)	Used to invoke **one-way** or **deferred response** asynchronous service requests
	Ticket invokeAsyncWithCallback(String, Object)	Used to invoke **request with callback** asynchronous service requests. The client must implement the ServiceCallback interface
	Object invokeResponse(Ticket, long)	Used to get response in the case of **deferred response** invocation
ServiceCallback	void onInvokeResponse(Ticket, Object, Exception)	Callback interface must be implemented by the client using a **request with callback** asynchronous service invocation

Exception handling for synchronous invocation:

When a service component is invoked synchronously, both the client and the service provider execute in the same thread. The target can return a response message, an exception, or nothing (in a one-way operation) to the client. If the result is an exception, it can be either a business exception or a system exception. The client in this case can be either application code or some form of system code.

Here is a sample client that invokes a Java component declared with a JType interface. The interface has one method declared as follows:

```
public interface StockQuote {
 float getQuote(String symbol) throws InvalidSymbolException;
}
```

The client code looks like this:

```
try {
 float quote = StockQuoteService.getQuote(String symbol);
 } catch (InvalidSymbolException s) {
 System.out.println(This is business exception declared in the Java interface.);
 } catch (ServiceRuntimeException e) {
 System.out.println(Unchecked system exception detected);
}
```

In the scenario, the first exception InvalidSymbolException indicates that the request has reached the service provider, which does not recognize the client input. The service provider then throws a business exception stating that the symbol supplied is invalid. This business exception is the only one declared by the method signature.

JType exceptions like InvalidSymbolException are only caught with clients using a JType reference.

In addition to the business exception declared, the client can receive system exceptions. For example, if the stock exchange system runs into a problem, the service might fail to obtain the quote with some unchecked exceptions. When such an exception is thrown by the service, a ServiceRuntimeException is returned to the client, and the client might then want to determine the underlying cause. The following code snippet shows how it can obtain this information:

```
try {
 float quote = StockQuoteService.getQuote(String symbol);
 } catch (ServiceRuntimeException e) {
 Throwable t = e.getCause();
 if (t instanceof RemoteException) {
 system.out.println(System ran into RemoteException. Details as follows:  + e.toString());
}
```

Exception handling for asynchronous invocation:

When a service component is invoked asynchronously, the client and service provider are executed in different threads, and error conditions can occur in either thread. The client may experience a system exception during the invocation, or the service provider may experience a business or system exception while servicing the request.

In WebSphere ESB, there is always an asynchronous counterpart of the synchronous interface.

Here is an example of an asynchronous interface:

```
public interface StockQuoteAsync {
public Ticket getQuoteAsync(String arg0);
public Ticket getQuoteAsyncWithCallback(String arg0);
public float getQuoteResponse(Ticket ticket, long timeout) throws InvalidSymbolException;
}
```

Here is the client code for a call using the invocation pattern for deferred response:

```
Ticket ticket = stockQuote.getQuoteAsync(symbol);
try {
 quote = stockQuote.getQuoteResponse(ticket, Service.WAIT);
 } catch (InvalidSymbolException s) {
```

```
 System.out.println(This is business exception declared in the interface.);
 } catch (ServiceRuntimeException e) {
 System.out.println(Unchecked system exception detected);
}
```

Like a synchronous invocation, InvalidSymbolException indicates that the request has reached the service provider, which has thrown a business exception stating that the symbol is invalid. This business exception is the only one declared by the interface. JType exceptions like InvalidSymbolException are caught only with clients using a JType reference.

In addition to the business exception declared, the client can receive system exceptions such as connection error that happens while sending the message. The client cannot receive system exceptions that happen in the service thread (the thread on the service side of the asynchronous invocation). According to the SCA asynchronous programming model, runtime exceptions that occur at the target component are not returned to the source component.

Exception case on asynchronous exception handling

There is one exception to the SCA asynchronous programming model rule that runtime exceptions that occur at the target component are not returned to the source component. If the source component is a Business Process component or Staff Process, system exceptions that occur in the target service component are returned to the caller. This capability lets business process designers model and catch system exceptions, and execute error logic if a BPEL client returns a system exception.

SCA data object containment and wrapper usages:

The SCA execution environment requires that invocation parameters be contained within a business object wrapper as determined by the operation and the interface.

Some interfaces require no wrapper when a single argument is accepted while other interfaces require a wrapper for any operation. For instance, a Java method which accepts a single parameter does not require the parameter be wrapped. All Document Literal Wrapped style WSDL operations require a wrapper regardless of the number of parameters.

When a wrapper is required, it can be instantiated by using the `BOFactory.createByType(...)` method as illustrated in the following example code:

```
Service service = (Service) locateService_servicePartner();
Type inputType = service.getReference().getOperationType("echoMulti").getInputType();
ServiceManager serviceManager = new ServiceManager();
BOFactory boFactory = (BOFactory) serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
DataObject wrapper = boFactory.createByType(inputType);
wrapper.set(0, new Integer(5));
wrapper.set(1, new Integer(10));
service.invoke("echoMulti", wrapper);
```

When no wrapper is required for single argument operations, the `OperationType.getInputType()` returns the type of the first and only argument. In this case, the `createByType` method can only be used when the Type describes a specific complex type. This method cannot be used for simple types or for abstract types such as `xs:anyType`.

Containment

Most business objects use a containment relationship which enforces a single parent relationship. Setting a business object value into the property of another business object moves the value to the new property, which results in removal from the original parent. You can see this relationship in action when you view the XML representation of the `DataObjects`:

```
<parent>
<child>
...
</child>
</parent>
```

In this example, "child" is always contained within the enclosing elements of its parent. It can be moved to another parent container, but cannot exist as a sub-element of two parents without copying the XML text `<child>...</child>`.

In order to avoid damaging the existing containment relationship, a `DataObject` must first be cloned before being set into another property. To prevent unnecessary cloning of `DataObject` parameters and facilitate a simplified programming model, the SCA container provides input and output types with non-containment properties when the interface is either a Java interface or a WSDL interface using the document literal wrapped style. No clone is necessary when the wrapper is created by using the Type returned by the `OperationType.getInputType()` or `OperationType.getOuputType()` methods for these interface types. To instantiate the wrapper for these types, use the `BOFactory.createByType(...)` method.

When the wrapper is created for other interface styles or by other Business Object Factory methods such as `createByElement`, the necessity of cloning `DataObject` parameters must be managed by the caller. Whether management of the containment relationship is necessary for the caller can be determined by introspecting the complex `Property` for the non-containment attribute: `Property.isContainment()`.

```
Type type = operationType.getInputType();
List properties = type.getProperties();
for(int i=0;i<properties.size();i++){
    Property property = (Property)properties.get(i);
    if(property.getType().getInstanceClass() == null || property.getType().getInstanceClass().equals(DataObject.class)){
        //Property is a complex type
        if(!property.isContainment()){
          //The property is a non-containment complex property. If a DataObject is set to this property, its parent will not be changed
        } else{
          //The property is a containment (ordinary) property. If a DataObject is set to this property, its previous parent will be discarded
        }

   } else{
        //The property is a simple type. All simple type properties are non-containment
   }
}
```

Regardless of whether a non-containment or containment wrapper type is passed when the invocation is made, the container can present the type found in the Operation metadata to downstream or upstream components. That is, the container prefers to use the type information as returned by the `OperationType.getInputType()` and `OperationType.getOutputType()` methods, and can replace another wrapper with the mentioned types as is appropriate.

When serialized, a non-containment property does not include the XML representation of the business object, but instead includes only a reference to the business object, which is expressed as a URI. Since this URI is likely not meaningful to the application, direct serialization of container provided non-containment business objects is not advised.

Object Reference Visibility

Data Object and Java Object parameters which are passed either directly or as a non-containment property of a wrapper are provided directly to component implementations when the invocation occurs on the same thread and within the same module. This means that the caller and the target both view the same business object or Java object, and therefore modifications made to the object in the target component be visible to the caller after the completion of the processing. The object reference is never passed between multiple threads or outside of a single module, so anytime the invocation is transferred to another thread or module, a copy of the original object is provided to the caller.

To promote component implementations which can be used in multiple wiring configurations, it is best to avoid expecting that any object modifications be visible upstream. Instead, if any modifications must be made visible to the caller, return the modified values. Doing so assures that the component can be used consistently regardless of how it is wired and invoked.

Considerations when invoking services on different servers:

One of the benefits of service-oriented architecture is the ability for consumers to use services that exist in other service modules. To balance the workload equitably, you should install applications on different servers in a cell and those applications should reside on different physical servers.

One of the advantages of Process Server is the ability to distribute the application workload across multiple servers in a cell. This distribution allows for better workload balancing among the various servers in the cell and maximizes the maintainability of the computing resources because there is only one copy of an application or service in the server. Thus, an application on server A could require a service installed in server B in the cell. To use services in this manner, you must configure communications between the servers. The type of configuration you perform depends on whether the calling service component invokes the service asynchronously or synchronously.

Related topics describe how to configure the systems for both asynchronous and synchronous invocations.

Configuring servers to invoke services asynchronously:

To enable service components on different servers to communicate, you have to configure the servers similarly. This topic describes the configuration you perform to enable the communication for applications that asynchronously invoke services on a different server.

Before you begin

The task assumes that you have already installed WebSphere ESB on the systems for which you are configuring the communications but have not yet installed the applications involved. You are using an administrative console that can examine and change the configuration for both servers involved.

About this task

Before installing an application that requires the services of a service component installed on another system, you must configure the systems so they can

communicate the requests. For service modules that use asynchronous invocations, the process involves foreign buses and Service Integration Bus (SIB) mediations.

Note: For the purposes of this task, the invoking service module resides on system A and the target resides on system B.

For the purposes of this task, Figure 262 contains the information to use in the configuration.

Figure 262. Invoking a service on a different system

Note: For simplicity, only the servers involved in this communication in each cell is shown and each server resides on a different physical machine.

Procedure

1. Collect information about each server involved in the communication. You need the following information for both the originator and target servers:
   - Host IP address
   - Cell
   - Node
   - Server

- Bus name
- Messaging engine
- Failed Event Queue name

2. Install the applications.
3. Create a foreign bus on each server pointing to the other server and set the routing definition type to `Direct, service integration bus link`.

   See the Connecting service integration buses to use point-to-point messaging topic in the WebSphere Application Server information center for more information.

   From the example, the configuration of the foreign bus and SIB mediation link on System A would be:

   ```
   Name of Service integration bus to connect to (the foreign bus):
   SCA.SYSTEM.ABCNode01Cell.Bus
   Gateway messaging engine in the foreign bus:
   ABCNode01.server1-SCA.SYSTEM.ABCNode01Cell.Bus
   Service integration bus link name: TestCrossCell
   Bootstrap service integration bus provider endpoints:
   9.26.237.144:7277:BootstrapBasicMessaging
   ```

   The configuration of the foreign bus and SIB mediation link on System B would be:

   ```
   Name of Service integration bus to connect to (the foreign bus):
   SCA.SYSTEM.WBINode01Cell.Bus
   Gateway messaging engine in the foreign bus:
   WPSNode.server1-SCA.SYSTEM.WBINode01Cell.Bus
   Service integration bus link name: TestCrossCell
   Bootstrap service integration bus provider endpoints:
   9.26.237.118:7276:BootstrapBasicMessaging
   ```

   Attention: The port number in the bootstrap is the SIB endpoint address port. If you enabled security, you must use the secure SIB endpoint address port.
4. Synchronize the SIB mediation links by restarting the servers.

   You should see messages like:

   ```
   [9/25/09 8:04:23:406 CDT] 00000034 SibMessage I [:] CWSIT0032I:
   The service integration bus link TestCrossCell from messaging engine
   WPSNode01.server1-SCA.SYSTEM.WPSNode01Cell.Bus in bus SCA.SYSTEM.WPSNode01Cell.Bus
   to messaging engine ABCNode01.server1-SCA.SYSTEM. ABCNode01Cell.Bus in bus SCA.SYSTEM.
   ABCNode01Cell.Bus started.
   ```
5. Display the destinations for each service module.
6. Modify the default forwarding path of outgoing destinations of the invoking service module that must be wired to targets on the other system.

   Select **Applications > SCA modules**, choose module and then click on **SCA system bus destinations**.

   The destination to wire has `importlink` in the destination name, for example on System A the destination would be `sca/AppA/importlink/test/sca/cros/simple/custinfo/CustomerInfo`. Modify the path by prefixing the foreign bus name to the destination name. From the example, the foreign bus name for the second system is `SCA.SYSTEM.ABCNode01Cell.Bus`. The result is

   ```
   SCA.SYSTEM.ABCNode01Cell.Bus:sca/AppA/importlink/
   test/sca/cros/simple/custinfo/CustomerInfo
   ```
7. Optional: Add sender roles to the foreign buses, if you enabled security on the systems. Make sure to define the user each application uses on both systems from the operating system command prompt. The command to add the role is:

   ```
   wsadmin $AdminTask addUserToForeignBusRole -bus busName
     -foreignBus foreignBusName -role roleName -user userName
   ```

Where:

busName
Is the name of the bus on the system you enter the command.

foreignBusName
Is the foreign bus to which you are adding the user.

userName
Is the userid to add to the foreign bus.

What to do next

Start the applications.

Configuring servers to invoke services synchronously:

When a service component invokes another service component synchronously, you must configure the invoking service component to point to the system running the target so the target service can communicate results to the invoking service component.

Before you begin

The task assumes that you have already installed WebSphere ESB on the systems for which you are configuring the communications but have not yet installed the applications involved. You are using an administrative console that can examine and change the configuration for both servers involved.

About this task

A service component invoking another service synchronously can communicate with the target only by configuring the export Java Naming and Directory Interface (JNDI) name on the target system to a JNDI name on the invoking system.

Note: For the purposes of this task, the invoking service module resides on system A and the target resides on system B.

For the purposes of this task, Figure 263 on page 730 contains the information to use in the configuration.

Figure 263. Invoking a service on a different system

Note: For simplicity, only the servers involved in this communication in each cell is shown and each server resides on a different physical machine.

Procedure

1. Install the applications on each server.
2. Create a new namespace binding on the invoking system (System A, in the example) pointing to the export on the target system.

   On the **Name Space Bindings** panel, select a scope of Cell and click **Apply**. With the changed scope, click **New** in the display to create the new binding.

   In the wizard, specify the following (the values are appropriate for the example configuration):

   a. Binding type is CORBA
   b. The basic properties are:
      - Binding identifier is a unique string, in our example: `sca_import_test_sca_cross_simple_custinfo_CustomerInfo`
      - Name in Name space is the JNDI name of the enterprise Java bean (EJB) you are invoking on the target system, for example:

        `sca/AppB/export/test/sca/cros/simple/custinfo/CustomerInfo`

This names the export interface on the target system.

- Corbaname URL is the IP address and port number of the naming service on the target system:

  `corbaname:iiop:`*host*`:`*port*`/NameServiceServerRoot#`*<package.qualified.interface>*

  Note: For IBM Process Server, the port is the `BOOTSTRAP_ADDRESS`.

  For example:

  ```
  corbaname:iiop:9.26.237.144:2809/NameServiceServerRoot#sca/
    AppB/export/test/sca/cros/simple/custinfo/CustomerInfo
  ```

When finished, click **Next** and verify the values on the **Summary** page. After verifying, click **Finish**.

Your system displays your new binding.

3. Save your changes by clicking **Save**.
4. If your cross cell configuration consists of servers on the same host with the same name and you encounter JNDI lookup failures with a `NameNotFoundException`, then you need to set a system property.

   Follow the directions in the Application access problems under the subheading *Two servers with the same name running on the same host are being used to interoperate*.

What to do next

Start the applications. The service component on System A can now synchronously invoke the service on System B.

Qualifiers

Qualifiers are an important part of SCA because they allow developers to place quality of service requirements on the WebSphere ESB runtime.

There are several different categories of qualifiers available in SCA. These categories of qualifiers are transaction, activity session, security, and asynchronous reliability.

Each SCA qualifier has a particular scope within the SCDL definition for a component where the qualifier can be specified. For example, some qualifiers can be specified at the references level, while others may only be valid at the interfaces or implementation level. The table lists the various qualifiers that are available and the valid scope for each. The qualifiers are sorted by the type of quality of service they provide, such as transaction or security.

Tip: See topics/aqosandtransactions.dita#aqosandtransactions for more detail.

Table 62. Summary of qualifiers

Type	Qualifier	Scope	Description
Transaction	transaction	Implementation	**global** – A global transaction must be present to run the component **local** – A global transaction must not exist to run the component **any** – Component is unaffected by transactional state **local application** – Component processing occurs within a WebSphere local transaction containment that is managed by the application
	joinTransaction	Interface	**true** – Hosting container joins client transaction **false** (default) – Hosting container does not join client transaction
	suspendTransaction	Reference	**true** – Synchronous invocations of target component do not run within client global transaction. **false** – Synchronous invocations of target component run within client global transaction
	deliverAsyncAt	Reference	**call** – Asynchronous invocations of a target service occur immediately **commit** – Asynchronous invocations of a target service occur as part of a global transaction

Table 62. Summary of qualifiers (continued)

Type	Qualifier	Scope	Description
Asynchronous Response	reliability	Reference	Specifies the quality of service level for asynchronous message delivery. Reliability can be one of the following values: **bestEffort** or **assured**
	requestExpiration	Reference	Specifies the length of time (milliseconds) after which an asynchronous request is to be discarded if not delivered
	responseExpiration	Reference	Specifies the duration (milliseconds) between the time a request is sent and the time a response or callback is received
Security	securityIdentity	Implementation	The **permission** specifies a logical name for the identity under which the implementation executes at runtime.
	securityPermission	Interfaces, Interface, Method	The caller identity must have the role specified from this qualifier in order to have permission to run the interface or method
Activity Session	activitySession	Implementation	**true** – There must be an ActivitySession established in order to run this component **false** – The component runs under no Activity Session**any** – The component does not depend on the presence or absence of an ActivitySession
	joinActivitySession	Interface	**true** – Hosting container joins client ActivitySession **false** – Hosting container does not join client ActivitySession
	suspendActivitySession	Reference	**true** – Methods on target component does NOT run as part of any client ActivitySession **false** – Methods on target component run as part of any client ActivitySession

Transaction qualifiers

Transaction qualifiers allow developers to request a particular transactional environment for the service components in an SCA module. The following is a summary of these qualifiers:

transaction
The transaction qualifier is set at the implementation scope of n service component. This qualifier can be set to either `'global'`, `'local'` (default), or `'any'`. When set to global, the component runs in the context of a global transaction. If a global transaction is present on invocation, the component is added to this global transaction scope. If set to local, the component runs in the context of a local transaction. Finally, if the value is set to any then if a global transaction is present the component joins the current global transaction scope. However, if a global transaction is not present, the component runs in the context of a local transaction.

joinTransaction
The joinTransaction qualifier is set at the interface scope of a service component. This qualifier can be set to either true or false (false being the default). If set to true, it instructs the runtime not to suspend a global transaction (if present) at the interface boundary. If set to false, it instructs the runtime to suspend a global transaction (if present) at the interface boundary. Exposing the joinTransaction transactional qualifier on an interface provides metadata that can be used by assemblers and deployers to ensure that the assembled application behaves as required. It is up to the assembler and deployer in addition to dynamic runtimes to reason about whether a target component federates with a propagated transaction.

suspendTransaction
The suspendTransaction qualifier is set at the reference level of a service component and identifies whether a global transaction should be suspended before invoking the target service associated with the reference. This qualifier can be set to either true or false (default).

deliveryAsyncAt
The deliveryAsyncAt qualifier is like the suspendTransaction qualifier, except that it pertains to asynchronous interactions rather than synchronous types as is the case with suspendTransaction. The deliverAsyncAt qualifier can have the value of call (default) or commit. If set to call, it indicates to the runtime that the message for the asynchronous interaction should be committed to the queue immediately when the call has been made. The value of commit indicates that the message should be committed to the queue as part of a transaction associated with the current unit of work.

Asynchronous response qualifiers

There are three qualifiers available for indicating quality of service for asynchronous response. Each of the asynchronous response qualifiers are specified at the reference scope. The following is a summary of asynchronous response qualifiers:

reliability
The reliability qualifier is used to specify the quality of service level for asynchronous message delivery. The reliability can be set to either bestEffort or assured (default).

requestExpiration
The requestExpiration qualifier is used to specify the length of time the runtime should hold onto an asynchronous request if it has not yet been delivered. After the time indicated for this qualifier, given in milliseconds, this request is discarded.

responseExpiration
The responseExpiration qualifier is used to specify the length of time that the runtime must retain an asynchronous response or must provide a callback. The value for this qualifier is given in milliseconds.

Security qualifiers

There are two qualifiers available for indicating quality of service related to security. The following is a summary of these qualifiers:

securityIdentity
The securityIdentity qualifier is used to specify the security identity under which the implementation for the service component should run at runtime. This qualifier must be placed at the implementation scope for the service component and the value given must match the logical name for the identity under which the component will run.

securityPermission
The securityPermission qualifier is specified at the interfaces level, including interface, or method level. The value for this qualifier indicates that a caller of this service must have the role that is specified in order to invoke the service.

For both the securityPermission and the securityIndentity, the underlying implementation for these qualifiers is based on existing Java EE concepts.

Activity session qualifiers

The set of activity session qualifiers are similar to the transaction qualifiers introduced earlier. The ActivitySession service is a WebSphere programming model extension that can provide an alternative unit of work when compared with global transactions. In fact, an activity session context can be longer lived than a global transaction and can even include global transactions. The following is a summary of the activity session qualifiers:

joinActivitySession
The joinActivitySession qualifier is set at the interface level, and indicates whether or not the component should join the activity session of a client caller. There are two values for this qualifier, true and false (default). If set to true it indicates that the runtime should not suspend an activity session if present when the component is invoked. If set to false it indicates that an activity session should be suspended before invoking the component.

activitySession
The activitySession qualifier is specified at the implementation level and is used to indicate whether or not an activity session should or should not exist in order to run the service component with which it is associated. This qualifier can be set to either ‘true', ‘false', or 'any‘ (default). If set to true, it indicates that the component will run as part of an activity session. If set to false, the component should not run as part of an activity session. This means that the joinActivitySession should also be set to false for any

interfaces specified for the component. Finally, if this qualifier is set to any, the component will run as part of an activity session if it is present, otherwise it will not.

suspendActivitySession
The suspendActivitySession qualifier is set at the reference level and is used to indicate whether or not a target service associated with a reference will get called as part of the calling activity session or not. If set to true, the activity session is suspended and the methods on the target component will not run as part of the client activity session. If set to false (default) the activity session is not suspended and methods on the target component will run as part of the client ActivitySession.

SCA programming techniques

This section provides examples of Service Component Architecture programming techniques.

Runtime rules used for Java to Service Data Objects conversion

To correctly override generated code, or to determine possible runtime exceptions related to Java to Service Data Object (SDO) conversions, an understanding of the rules involved is important. The majority of the conversions are straightforward, but there are some complex cases where the runtime provides the best possibility when it converts the generated code.

Basic types and classes

The runtime performs a straightforward conversion between Service Data Objects and basic Java types and classes. Basic types and classes include:
- Char or java.lang.Character
- Boolean
- Java.lang.Boolean
- Byte or java.lang.Byte
- Short or java.lang.Short
- Int or java.lang.Integer
- Long or java.lang.Long
- Float or java.lang.Float
- Double or java.lang.Double
- Java.lang.String
- Java.math.BigInteger
- Java.math.BigDecimal
- Java.util.Calendar
- Java.util.Date
- Java.xml.namespace.QName
- Java.net.URI
- Byte[]

User-defined Java classes and arrays

When converting from a Java class or array to an SDO, the runtime creates a data object that has a URI that is generated by inverting the package name of the Java type and has a type equal to the name of the Java class. For example, the Java class `com.ibm.xsd.Customer` is converted to an SDO and URI `http://xsd.ibm.com` with type Customer. The runtime then inspects the contents of the Java class members and assigns the values to properties in the SDO.

When converting from an SDO to a Java type, the runtime generates the package name by inverting the URI and the name of the type equals the type of the SDO. For example, the data object with type Customer and URI `http://xsd.ibm.com` generates an instance of the Java package `com.ibm.xsd.Customer`. The runtime then extracts values from the properties of the SDO and assign those properties to fields in the instance of the Java class.

When the Java class is a user-defined interface, you must override the generated code and provide a concrete class that the runtime can instantiate. If the runtime cannot create the concrete class, an exception occurs.

Java.lang.Object

When a Java type is java.lang.Object the generated type is xsd:anyType. A module can invoke this interface with any SDO. The runtime attempts to instantiate a concrete class the same way it does for user-defined Java classes and arrays, if the runtime can find that class. Otherwise, the runtime passes the SDO to the Java interface.

Even if the method returns a java.lang.Object type, the runtime converts to an SDO only if the method returns a concrete type. The runtime uses a similar conversion to that for converting user-defined Java classes and arrays to SDOs, as described by the next paragraph.

When converting from a Java class or array to an SDO, the runtime creates a data object that has a URI that is generated by inverting the package name of the Java type and has a type equal to the name of the Java class. For example, the Java class `com.ibm.xsd.Customer` is converted to an SDO and URI `http://xsd.ibm.com` with type Customer. The runtime then inspects the contents of the Java class members and assigns the values to properties in the SDO.

In either case, if the runtime is unable to complete the conversion an exception occurs.

Java.util container classes

When converting to a concrete Java container class such as Vector, HashMap, HashSet and the like, the runtime instantiates the appropriate container class. The runtime uses a method similar to that used for user-defined Java classes and arrays to populate the container class. If the runtime cannot locate a concrete Java class, the runtime populates the container class with the SDO.

When converting concrete Java container classes to SDOs, the runtime uses the generated schemas shown in “Java to XML conversion.”

Java.util interfaces

For certain container interfaces in the `java.util` package, the runtime instantiates the following concrete classes:

Table 63. WSDL type to Java class conversion

Interface	Default concrete classes
Collection	HashSet
Map	HashMap
List	ArrayList

Table 63. WSDL type to Java class conversion (continued)

Interface	Default concrete classes
Set	HashSet

Overriding a Service Data Object to Java conversion

Sometimes, the conversion the system creates between a Service Data Object (SDO) and a Java type object may not meet your needs. Use this procedure to replace the default implementation with your own.

Before you begin

Make sure that you have generated the WSDL to Java type conversion using either IBM Integration Designer or the **genMapper** command.

About this task

You override a generated component that maps a WSDL type to a Java type by replacing the generated code with code that meets your needs. Consider using your own map if you have defined your own Java classes. Use this procedure to make the changes.

Procedure

1. Locate the generated component. The component is named *java_class*`Mapper.component`.
2. Edit the component using a text editor.
3. Comment out the generated code and provide your own method.

   Do not change the file name that contains the component implementation.

Example

This is an example of a generated component to replace:

```
 private Object datatojava_get_customerAcct(DataObject myCustomerID,
    String integer)
{

  // You can override this code for custom mapping.
  // Comment out this code and write custom code.

  // You can also change the Java type that is passed to the
  // converter, which the converter tries to create.


  return SDOJavaObjectMediator.data2Java(customerID, integer) ;

}
```

What to do next

Copy the component and other files to the directory in which the containing module resides, and either wire the component in IBM Integration Designer or generate an enterprise archive (EAR) file using the **serviceDeploy** command.

Overriding the generated Service Component Architecture implementation

Sometimes, the conversion the system creates between a Java code and a Service Data Object (SDO) may not meet your needs. Use this procedure to replace the default Service Component Architecture (SCA) implementation with your own.

Before you begin

Make sure that you have generated the Java to Web Services Definition Language (WSDL) type conversion using either IBM Integration Designer or the **genMapper** command.

About this task

You override a generated component that maps a Java type to a WSDL type by replacing the generated code with code that meets your needs. Consider using your own map if you have defined your own Java classes. Use this procedure to make the changes.

Procedure

1. Locate the generated component. The component is named *java_class*`Mapper.component`.
2. Edit the component using a text editor.
3. Comment out the generated code and provide your own method.

   Do not change the file name that contains the component implementation.

Example

This is an example of a generated component to replace:

```
private DataObject javatodata_setAccount_output(Object myAccount) {

  // You can override this code for custom mapping.
  // Comment out this code and write custom code.

  // You can also change the Java type that is passed to the
  // converter, which the converter tries to create.


return SDOJavaObjectMediator.java2Data(myAccount);


}
```

What to do next

Copy the component and other files to the directory in which the containing module resides, and either wire the component in IBM Integration Designer or generate an enterprise archive (EAR) file using the **serviceDeploy** command.

Protocol header propagation from non-SCA export bindings

The context service is responsible for propagating the context (including the protocol headers, such as the JMS header, and the user context, such as account ID) along a Service Component Architecture (SCA) invocation path. The context service offers a set of APIs and configurable settings.

When the context service propagation is bi-directional, the response context always overwrites the current context. When you are running an invocation from one SCA component to another, a response contains a different context. A service component has an incoming context, but when you invoke another service, the other service overwrites the original outgoing context. The response context becomes the new context.

When the context service propagation is one way, the original context remains the same.

The lifecycle of the context service is associated with an invocation. A request has associated context, and the lifecycle of that context is bound to the processing of that particular request. When that request is finished processing, then the lifecycle of that context ends.

Context services have configurable rules and tables that dictate the binding behavior. For more information, see the Generated API and SPI documentation that is available in the Reference section. During development in the IBM Integration Designer, you can set the context service on import-export properties. For more details, see the import and export bindings information in the IBM Integration Designer information center.

Business objects programming

Business objects are containers for application data, such as a customer or an invoice. Data is exchanged between components by way of business objects.

The underlying structure of a business object is an XML schema definition (XSD), and programmatic access to business objects is provided via business object interfaces in WebSphere. Collectively, these aspects of the business object, its structural representation, its programmatic interfaces, and its behavior and manipulation within the service component architecture (SCA), are the business object framework, which provides a powerful, consistent means for describing and delivering business data in your solution.

Programming model

The business object programming model section describes how the basic types of data are encapsulated within the IBM business object framework. To facilitate the creation and manipulation of business objects, the business object framework extends Service Data Objects specifications by providing a set of Java services.

Working with the IBM business object framework

The business object framework describes how data in the run time is modeled by applications, integrated into the run time, and represented in memory.

Table 1 summarizes how the basic types of data are implemented in the business object framework.

Table 64. Data abstractions and the corresponding implementations

Data Abstraction	Implementation	Description
Instance data	Business object	Business objects are the primary mechanism for representing business entities, or document literal message definitions, enabling everything from a simple basic object with scalar properties to a large complex hierarchy or graph of objects. A business object is a direct corollary to the SDO DataObject concept.
Instance metadata	Business graph	Business graphs are wrappers that are added around a simple business object or a hierarchy of business objects to provide additional capabilities, such as carrying change summary and event summary information related to the business objects in the business graph. A business graph is a direct corollary of the SDO DataGraph concept, except that it provides more than just the single change summary header.
Type metadata	Enterprise metadata Business object type metadata	Business object type metadata is the metadata that can be added to business object definitions to enhance their value in the runtime. These metadata items are added to the business object's XML schema definition as well known `xs:annotation` and `xs:appinfo` elements.
Services	Business object services (APIs)	Business object services are a set of capabilities provided on top of the basic capabilities provided by Service Data Objects. Examples are services such as create, copy, equality, and serialization. These APIs are found in the `com.ibm.websphere.bo` package.

The WebSphere ESB business object framework is an extension of the SDO standard. Therefore, business objects exchanged between WebSphere ESB components are instances of the `commonj.sdo.DataObject` class. However, the WebSphere ESB business object framework adds several services and functions that simplify and enrich the basic DataObject functionality.

To facilitate the creation and manipulation of business objects, the WebSphere business object framework extends SDO specifications by providing a set of Java services. These services are part of the package named `com.ibm.websphere.bo`.

- **BOFactory**: The key service that provides various ways to create instances of business objects.
- **BOXMLSerializer**: Provides ways to "inflate" a business object from a stream or to write the content of a business object, in XML format, to a stream.
- **BOCopy**: Provides methods that make copies of business objects ("deep" and "shallow" semantics).

- **BODataObject**: Gives you access to the data object aspects of a business object, such as the change summary, the business graph, and the event summary.
- **BOXMLDocument**: The front end to the service that lets you manipulate the business object as an XML document.
- **BOChangeSummary** and **BOEventSummary**: Simplifies access to and manipulation of the change summary and event summary portion of a business object.
- **BOEquality**: A service that enables you to determine whether two business objects contain the same information. It supports both deep and shallow equality.
- **BOType** and **BOTypeMetaData**: These services materialize instances of the `commonj.sdo` Type and let you manipulate the associated metadata. Instances of Type can then be used to create business objects "by type".
- **BOInstanceValidator**: Validates the data in a business object to see if it conforms to the XSD.

Modeling business objects

Business object data that flows through the WebSphere ESB runtime is modeled using XML schemas. XML schemas are an alternative to document type definitions (DTDs) and can be used to extend functionality in the areas of data typing, inheritance, and presentation. XML schema provides a form for modeling the data types that is industry standard, widely adopted, and is platform and language neutral.

Target namespace definition:

Most business and communications problems that XML can solve require a combination of several XML vocabularies. XML has a mechanism for qualifying names to be allocated into different namespaces, such as namespaces that apply to different industries. In XML, a uniform resource identifier (URI) provides a unique name to associate with the element, attribute, and type definitions in an XML schema.

There are two requirements for the business object target namespace:

- The business process management run time and tools prefer target namespaces to look like `http://www.foo.com/xyz` versus `urn:foo:com:xyz`.
- The business object framework requires a target namespace for business objects.

Business object definition:

WebSphere ESB provides a flexible mechanism for defining or importing business objects.

There are essentially three different forms of XML schema that WebSphere ESB recognizes as a business object definition:

- A top-level complex type definition
- A top-level anonymous complex type definition
- A top-level element that references a named complex type

Top-level complex type definition

This is an example of a top-level complex type definition:

```
<complexType name="ProductType">
 <sequence>
  <element name="name" type="string"/>
  <element name="color" type="string" maxOccurs="unbounded"/>
 </sequence>
</complexType>
```

Importing or defining business objects that are all defined using complex type definitions only is the most flexible and manageable scheme. The upside of this model is a type library that enables reuse. Reuse can be by three different methods.

- First, new types can be created using the complex type derivation model (of extension or restriction).
- Second, new aggregate types can be created using the existing complex types and available simple types as primitives.
- Third, new complex document definitions can be created, like the aggregate complex types.

The other implication of business objects defined as complex types is that when the complex type is used by the `JService` component kinds to transfer data within the runtime, in order to maintain WS-I compliance, an element needs to be created that references the named complex type.

Top-level anonymous complex type definition

This is an example of a top-level anonymous complex type definition:

```
<element name="Product">
 <complexType>
  <sequence>
   <element name="name" type="string"/>
   <element name="color" type="string" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

If the imported business objects are all anonymous element definitions, they are ready made to be included in `JService` invocations. However, they are not inherently reusable.

Top-level element referencing a named type

This is an example of a top-level element referencing a named type:

```
<element name="product" type="prod:ProdType"/>
```

Business objects that reference named complex types might be frequent in an environment that has already defined WSDL operations that require element definitions. In this scenario, it is important to consider the possible disposition of the complex type and element definitions:

- Elements can cohabitate with their complex type definitions in the same XML schema file.
- Elements can cohabitate with their complex type definitions embedded in a WSDL file.
- Elements can be defined in XML schema `A.xsd`, while their complex type definition is defined in XML schema file `B.xsd`.
- Elements can be embedded in a WSDL file, referencing a complex type definition defined in an XML schema file.

Example

The example demonstrates all the mechanisms for defining a business object combined together.

```
<schema
 targetNamespace="http://www.app.com/Address"
 xmlns:addr="http://www.app.com/Address"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">

 <complexType name="Address">
  <sequence>
   <element name="street1" type="string"/>
   <element name="street2" type="string"/>
   <element name="city" type="string"/>
   <element name="state" type="string"/>
   <element name="zip" type="string"/>
  </sequence>
 </complexType>

 <element name="homeAddress" type="addr:Address"/>
 <element name="workAddress" type="addr:Address"/>
 <element name="otherAddress" type="addr:Address"/>

 <element name="individualContact">
  <complexType>
   <sequence>
    <element name="firstName" type="string"/>
    <element name="lastName" type="string"/>
    <element ref="addr:HomeAddress"/>
    <element ref="addr:WorkAddress"/>
    <element ref="addr:OtherAddress"/>
   </sequence>
  </complexType>
 </element>

 <element name="businessContact">
  <complexType>
   <sequence>
    <element name="name" type="string"/>
    <element ref="addr:WorkAddress"/>
   </sequence>
  </complexType>
 </element>

 <element name="chairmanOfTheBoard">
  <complexType>
   <sequence>
    <element name="startDate" type="date"/>
    <element ref="addr:IndividualContact"/>
    <element ref="addr:BusinessContact"/>
   </sequence>
  </complexType>
 </element>
</schema>
```

The following guidelines are the preferred way to define business objects:

- Elements are defined using named types; anonymous types are discouraged.

- Elements and complex type definitions do not cohabitate the same XML schema or WSDL file. This practice discourages type reuse.
- Complex types are defined in XML schema files, not WSDL definitions, creating a type library like concept. Again, this type of definition enables and encourages complex type reuse.
- Element definitions are built as necessary to reference a single complex type definition. For example, the definition of an element inside the WSDL is a pattern that is encouraged.
- Element definitions typically use the same target namespace as their complex type definition.

Business object property definition:

XML schema provides complex types, simple types, and attributes, which are used to build business objects.

Complex type definitions, anonymous complex type definitions, and elements referencing complex type definitions are used to define the outer business objects. The term property is used to define the data inside a business object. The term is derived from the Service Data Object term property, and is defined by the `commonj.sdo.Property` interface. It is synonymous with the concept of an attribute.

A property can either be simple or complex. A simple property can be defined either as an XML schema attribute, or as an XML schema element with a type that is an XML schema simple type. A complex property can either reference another business object, or it can define a complex structure within the current business object.

The full XML schema type system is supported.

Figure 264. XML schema simple types

Supported XSD and WSDL artifacts:

When a WSDL or a schema is imported into a project in IBM Integration Designer, the business objects rendered from the WSDL or schema can then be used to develop a module. It is important to note however, that only certain artifacts from a schema are rendered as *business objects* (for example, root/top level elements and named complex types). Certain artifacts, such as nested anonymous complex types, are *not* rendered as business objects. These restrictions are a result of which artifacts are accessible in the XML schema. For example, if you import a schema which resulted in only one business object, it is most likely that the rest of the elements were anonymous complex types. The following information details which XSD and WSDL artifacts result in business objects.

Business objects from imported XSD definitions

When an XML schema is imported into a project, only certain artifacts are rendered as business objects. The following lists show which artifacts are supported at authoring time and at runtime:

XSD artifacts resulting in business objects at authoring time:

- Complex types defined at the root level
- Elements defined at the root level with anonymous complex types

These artifacts result in user-defined simple types at authoring time which can be referenced by business objects:

- Simple types defined at the root level
- Elements defined at the root level with anonymous simple types

Business objects from imported WSDL files

When a WSDL definition that includes an inline XSD schema is imported into a project, only certain artifacts are rendered as business objects. The following lists show which artifacts are supported at authoring time and at runtime:

Inline XSD artifacts resulting in business objects at authoring time:

- Complex types defined at the root level
- Elements defined at the root level with anonymous complex types AND the name of the element does not contain the names of any operations/messages (as these elements could be doc-lit-wrapped elements which IBM Integration Designer unwraps automatically)

These artifacts result in user-defined simple types at authoring time which can be referenced by business objects:

- Simple types defined at the root level
- Elements defined at the root level with anonymous simple types

Runtime business objects from XSD artifacts

These artifacts result in business objects at runtime:

- Complex types defined at the root level
- Elements defined at the root level with anonymous complex types
- Elements defined at the root level which reference a complex type

Runtime business objects from WSDL files

These artifacts result in business objects at runtime:

- Complex types defined at the root level
- Elements defined at the root level with anonymous complex types AND the name of the element does not contain the names of any operations/messages (as these elements could be doc-lit-wrapped elements which IBM Integration Designer unwraps automatically)
- Elements defined at the root level which reference a complex type

Flat and hierarchical business objects:

Business objects can be modeled as flat or as hierarchical.

Flat business object

A *flat business object* contains one or more simple attributes and a list of supported verbs. A simple attribute represents one value, such as a String or Integer or Date. All simple attributes have single cardinality. If the business object is an application-specific business object, a flat business object can represent one entity in an application or in a technology standard.

Hierarchical business object

Hierarchical business object definitions define the structure of multiple related entities, encapsulating not only each individual entity but also aspects of the relationship between entities. In addition to containing at least one simple attribute, a hierarchical business object has one or more attributes that are complex (that is, the attribute itself contains one or more business objects, called *child business objects*). The business object that contains the complex attribute is called the *parent business object*.

There are two types of relationships between parent and child business objects:

- Single cardinality - When an attribute in a parent business object represents a single child business object. The type of the attribute is set to the name of the child business object, and the cardinality is set to one.
- Multiple cardinality - When an attribute in the parent business object represents an array of child business objects. The type of the attribute is set to the name of the child business object, and the cardinality is set to *n*.

In turn, each child business object can contain attributes that contain a child business object, or an array of business objects, and so on. The business object at the top of the hierarchy, which itself does not have a parent, is called the *top-level business object*. Any single business object, independent of its child business objects that it might contain (or that might contain it), is called an *individual business object*.

Example

The following example helps illustrate the difference between a flat business object and hierarchical business object. The diagram contains a flat business object, named Product. The business object is represented in memory with the Service Data Object type `commonj.sdo.DataObject` (unless it was statically generated). This flat business object has a set of attributes that are modeled as XML schema simple types as well as an attribute that is modeled as a list of simple types.

The diagram also illustrates a Product business object, in combination with the ProductCategory business object, to create a more complex hierarchical business object. This business object has both a top-level business object (ProductCategory), as well as a contained business object (Product).

Figure 265. Comparison of flat and hierarchical business objects

Here is example of the flat business object definition for the Product business object. The Product business object defines two properties, *Name* and *Inventory*, that are typed by the XML schema simple types `xs:string` and `xs:int`. In addition, Product also demonstrates the definition of List property *Color*, which is a List of `xs:string` simple types.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">

 <complexType name="Product">
  <sequence>
   <element name="name" type="string"/>
   <element name="inventory" type="int"/>
   <element name="color" type="string" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</schema>
```

Here is example of the hierarchical business object ProductCategory. The definition defines two different business objects, ProductCategory and Product. The hierarchical ProductCategory business object defines the property, *Name* and also defines a List of business objects of type `Product` or `ProductCategory`.

```
<schema>
 targetNamespace="http://www.scm.com/ProductCategoryTypes"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">
 elementFormDefault="qualified">

 <complexType name="ProductCategory">
  <sequence>
   <element name="name" type="string"/>
   <choice>
    <element name=="productCategory"
     type="pc:ProductCategory"
     maxOccurs="unbounded"/>
    <element name=="product"
```

```
      type="pc:Product"
      maxOccurs="unbounded"/>
    </choice>
   </sequence>
  </complexType>

  <complexType name="Product">
   <sequence>
    <element name="name" type="string"/>
    <element name=="inventory" type="int"/>
    <element name=="color" type="string" maxOccurs="unbounded"/>
   </sequence>
  </complexType>

</schema>
```

Business object characteristics:

Business objects have inherent characteristics that enhance their use within the business object framework.

Cardinality

The cardinality of properties is defined by standard XML schema `minOccurs` and `maxOccurs` facets for simple and complex types and the `use` attribute for attributes.

Default property values

The capability to provide default values in XML schema for attributes and simple types in a business object is supported by the business object framework. This support is visible at creation time when the simple property types of a business object reflect their default values.

Nillable

An element can be defined in XML schema to be nillable. The business object framework enables properties that are *nillable* to have their value set to `Null` at runtime.

Key definition

Business object key information can be used by multiple subsystems, such as relationship, sequencing, and isolation. However, each of these subsystems can define their own key mechanism independent of the business object's key definition. Since the underlying model language leveraged by business objects is XML schema, first class support for key definitions exists within the modeling language. However, this support within the modeling language is not fully supported in the SDO runtime.

xs:ID, xs:IDREF, and xs:IDREFS

These types were added to XML schema primarily to provide an upgrade path for DTDs. Each complex type can have 0 or 1 elements/attributes typed as an `xs:ID`. IDs must be unique to an entire document, as opposed to a primary key in a database, for example, that must be unique with respect to the scope of the table. As an example, a conformant document cannot use the same ID value to identify both a Product and a ProductCategory. Often elements get around this restriction by prepending the complex type name to the key value. An attribute typed `IDREF` must

contain a value that matches one of the ID values in the current document. In addition, XML schema provides for a construct which is an element that can be typed to contain a list of ID references, `xs:IDREFs`.

xs:unique, xs:key, xs:keyref
XML schema introduced a new style enabling key definitions and key references. The `xs:unique` construct enables a user to define that 1 or more fields in an element that must be unique within a particular scope of the element (which represent the entire document). The `xs:key` construct is a variant of `xs:unique` with the additional constraint that the elements referenced are required. The `xs:keyref` construct is used to identify that the value of an element must be named key or unique construct.

The `unique`, `key`, and `keyref` constructs have several advantages over the `ID`, `IDREF`, and `IDREFS` set, including:

- They can define compound keys.
- They can define unique constraints that are relative to a portion of the document.

Although a business object is not required to have a defined key, it is highly recommended. Business objects that do not define a key can be used by applications. This scenario is a common use model in many Java EE centric applications use models, where JavaBeans are passed back and forth between the servlet and EJB containers without the specification of a key. However, those business objects that do not define a key are unable to interact with the subsystems that require a key. This situation limits their ability to take advantage of WebSphere ESB qualities of service.

Modeling business graphs

Business graphs relate to business objects in much the same way SDO DataGraphs relate to SDO DataObjects. When a top-level business object requires enrichment to be able to use the services provided by WebSphere ESB, it is wrapped with a business graph. The business graph wrapper provides the additional value add by adding data headers for storing information logically in memory, or physically when the business graph is serialized.

Note: If you are migrating an application from WebSphere InterChange Server or migrating adapters, you may need to use business graphs.

Business graph use models:

Two primary use models represent the fundamental capabilities provided by the business graph: delta support and after image.

Delta support is the capability enabled by the SDO 1.0, where changes to a business object graph are captured in a special header called the *change summary*.

An *after image* is a business graph that captures the current state of business data in an EIS system, typically as a result of a change to that data in the EIS. An after image enables changes in EIS systems to be captured and published to the runtime.

To provide these two fundamental concepts, business graphs introduce and provide for the following concepts:

- *Templated business graph* is a business graph that is typed specifically for a type of business object graph that it wraps.

- *Change summary* is provided to capture implicit changes, explicit changes, for both Delta, as well as After Image use models.
- *Explicit change summary* support is provided by the business graph programming interfaces enabling BPM component, such as adapters, mediators, maps, and relationships to explicitly modify the Change Summary header.
- *Event summary* is provided to support capturing instance-based annotations about data in the business object graph, and specifically, to contain object event identifiers
- *Verb* support is provided by the business graph enabling the components in the runtime to key off the event type to perform value added function.
- *Supported verbs* is the notion of constraining and extending the set of allowable verbs that can be specified for a business graph.
- *Object event identifiers* are supported by the business graph enabling all the objects in a graph to be uniquely identified. This capability is required by some of the components that provide value added features.

Business graph model definition:

To remain non-intrusive to an externally developed model of a business object, the business graph capability is wrapped around the original business object. A pattern named the templated business graph is used to wrap the original business object with the enriched business graph schema.

The *templated business graph* is created by extending the business graph complex type that is provided by the business object framework runtime and adding an element delegating to the original business object. The diagram shows a UML model for the business graph. The business graph is abstract, providing just the standard set of headers that are added to the top-level business object.

Figure 266. Business graph complex type

The XML schema model for the abstract business graph XML schema complex type:

```
<schema
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:sdo="commonj.sdo"
```

```
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="commonj.sdo" schemaLocation="DataGraph.xsd"/>

 <complexType name="BusinessGraph" abstract="true">
  <sequence>
   <element name="changeSummary" type="sdo:ChangeSummaryType"
    minOccurs="0" maxOccurs="1"/>
   <element name="eventSummary" type="bo:EventSummary"
    minOccurs="0" maxOccurs="1"/>
   <element name="property" type="bo:ValuesType"
    minOccurs="0"/>
  </sequence>
  <anyAttribute namespace="##other" processContents="lax"/>
 </complexType>

 <complexType name="EventSummary">
  <sequence>
   <any namespace="##any" processContents="lax"
    minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
 </complexType>

 <complexType name="ValuesType">
  <complexContent>
   <extension base="ecore:EClass"/>
  </complexContent>
 </complexType>

 <attribute name="name" type="string"/>
</schema>
```

Business graphs are only a top-level concept because they exist to add a set of headers to an existing top-level business object, and cannot be modeled in a recursive design pattern like business objects can. Business graphs can be applied to any business object, but upon application, that business object becomes a top-level business object.

This is an example of a business graph that wraps a business object named ProductCategory, ProductCategory is a hierarchical business object that contains a child business object named Product.

```
<schema
 targetNamespace="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pcbg="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
  schemaLocation="BusinessGraph.xsd"/>

 <import namespace="http://www.scm.com/ProductCategoryTypes"
  schemaLocation="ProductCategoryTypes.xsd"/>

 <complexType name="ProductCategoryBG">
```

```
  <complexContent>
   <extension base="bo:BusinessGraph">
    <sequence>
     <element name="verb" minOccurs="0" maxOccurs="1"/>
     <element name="productCategory"
      type="pc:ProductCategory"
      minOccurs="0" maxOccurs="1"/>
    </sequence>
   </extension>
  <complexContent>
 <complexType>
</schema>
```

The recommended pattern for a generated templated business graph is:

- The templated business graph are defined using a named complex type that extends the `bo:BusinessGraph` schema by restriction (this pattern provides restrictions on the allowable Verb value).
- The name of the templated business graph is the name of the top-level business object, with the string "BG" appended.
- The target namespace of the templated business graph is composed of the target namespace of the business object being wrapped, followed by a "/", named by the complex type of the templated business graph.
- The templated business graph complex type definition are placed in its own XML schema file whose name corresponds to the name of the complex type.

Business graph model instance:

The top-level business graph is represented in memory much the same as a business object, by an SDO 1.0 DataObject, specifically, the class `commonj.sdo.DataObject`.

The business graph is composed of more than just the business graph containment object. It also contains two headers, plus the top-level business object. None of the headers is represented in memory as a DataObject, and neither of the headers enables a DataObject API access mechanism.

Change summary header

A capability provided by business graphs is the ability to implicitly track changes to the business object in the business graph as the business graph is passed between several different business processes. As each process changes the business graph, a change log is generated in memory. Once the business graph is serialized, the change log is written out in a format that enables the next process to see the types of changes that have been made to the business graph. This technique enables adapter and Data Mediator services to efficiently update their persistence data stores by optimizing the data they have to focus on.

In addition, the change summary is also used when an adapter generates an After Image event to describe data that was updated in an EIS system. In particular, the ability of the change summary to annotate object and property changes is used in the After Image use model (as well as the verb for the business graph).

Implicit change summary usage

When applications cause changes to the business object, the applications can turn on change logging so change events are automatically logged to the change

summary. The change summary is included in the business graph at a peer level of the top-level business object. Change events are defined at two levels, for business objects, and for business object properties. Business objects have create, update, and delete change types, while properties have set and unset change types. Change events are only tracked for modifications to the business object portion of the business graph. A change in the event summary does not result in an implicit update of the change summary for the business graph.

Explicit change summary modification

There are several use models in the WebSphere ESB components that require the ability to explicitly write to the change summary header. For example, an adapter that generates an EIS event explicitly creates the object change types, and potentially the property change types. An application-specific business object/general business object (ASBO/GBO) map transforms the change summary from one business graph to another, creating a new version of the change summary in an output business graph. This capability is provided by the business object framework.

Event summary header

The event summary provides the **ObjectEventID**, which is the mechanism used to uniquely identify an instance of an object that appears in the runtime. This information is carried in the event summary, where the unique identifier is associated with a given DataObject in the business object hierarchy of the business graph.

Event information can also be carried in the event summary. This information is a string that can be used to add additional metadata associated with each object in the business object hierarchy for the business graph. One potential use model for event information is to mark up contained business objects with a verb other than the standard Create, Update, and Delete verbs supported by change summary.

Verb header

If the verb is set on the business graph, then the business object data portion of the business graph carries an EIS After Image data set. If the verb contains a value, there are three possibilities with respect to the granularity of the after image event:

- The change summary is empty. This situation occurs when an EIS knows about the type of update to a set of data, but does not have specifics on which objects in the graph have been created, updated, or deleted. The result is that a downstream mediation, map, relationship, or adapter, must use the information in the business graph plus additional data to determine the actual update to perform.
- The change summary has object level change event annotations. This case is typical where the EIS system recognizes what has happened to each object in the business graph, but lacks the granularity to determine whether specific properties of the objects have been updated.
- The change summary has object level change event annotations and property level get/set annotations. This situation is the most granular case, and all adapters strive to obtain this level of an After Image if their EIS system makes the appropriate data available. The advantage of a fully specified After Image is that it enables property level implicit property change event management to occur. Therefore, it much easier for After Image business graphs to interoperate with the disconnected Delta-based Business Graphs.

Modeling business object type metadata

Business object type metadata can be added to business object definitions to enhance their value in the runtime. The business object framework allows you to design, annotate, and convert business object type metadata.

The business object framework provides a mechanism that enables:

- Metadata to be mixed into a business object in a consistent, and relatively non-intrusive fashion
- A prescriptive policy for developers to define complex annotation structures
- A prescriptive policy for business object developers and deployers to annotate a business object with instance metadata that conforms to the predefined complex annotation structures
- A set of APIs for transforming the annotations at runtime into a simple to use DataObject structure

This process involves at least three different roles:

- The first role is that of the business object type metadata designer. This role designs the metadata structure. For example, the business object framework plays this role and defines a couple of metadata characteristics for annotating a business object. It is expected that adapters such as PeopleSoft, Siebel, and SAP, play the role of the metadata designer to annotate the business object with application-specific information.
- The second role is the business object designer or deployer. This role uses the structure of the business object type metadata, and the policy defined by the business object type metadata framework, to annotate the business object definitions with metadata.
- If the business object is annotated correctly, the third role can use the business object metadata APIs to validate and inspect the metadata at runtime, and turn it into a navigable and useful DataObject graph structure.

In addition, the business object framework provides for the following metadata features:

- Compound primary key and foreign key property definitions
- Top-level supported verb metadata annotations

For more details, refer to these other topics.

Representing business object type metadata:

The business object framework defines a mechanism by which business object type metadata can be mixed into either a top down, or externally developed, imported schema.

The mechanism that is used to mix in the business object type metadata, is XML schema annotations and the `appInfo` structure. This policy, although it requires modification of the original schema definition, does so in a way that enables the annotations and `appinfo` to be easily view toggled, or removed completely. This identification scheme is done by using the source attribute on the `xs:appinfo` tag whose value is the target namespace that defines the business object type metadata.

For example, assume that the following schema was imported into the runtime. The schema describes a business object, named Product, that includes customer annotations.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:p="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified

 <complexType name="Product">
  <annotation>
   <appInfo>
    <SCMEditor value="Bottom" type="Anchor"/>
   </appInfo>
   <documentation>
    Describes the SCM Product
   </documentation>
  </annotation>

  <sequence>
   <element name="id" type="ID"/>
   <element name="description" type="string" default="DefaultDescription"/>
   <element name="sku" type="p:Sku"/>
  </sequence>
 </complexType>

 <simpleType name="Sku">
  <restriction base="string">
   <pattern value="\d{3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>
</schema>
```

Designing business object type metadata:

You can design a metadata structure to house the business object metadata.

About this task

To design the metadata structure follow these steps.

Procedure

1. Create an XML schema file with a valid target namespace. This step is used when the instance metadata is added to the business object definition.
2. Define a named complex type to define each separate piece of metadata that can be added to a business object. The complex type definition is used to define the structure of the dynamically typed `DataObject` that is used to read the instance metadata. The name of the complex type is used when the instance metadata is added to the business object definition.

Example

This example shows a portion of business object type metadata developed for a PeopleSoft adapter.

```
<schema>
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/adapter/psft/
  PSFTBODefinitionASI/7.0.0"
 xmlns:psft="http://www.ibm.com/xmlns/prod/websphere/adapter/psft/
  PSFTBODefinitionASI/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <complexType name="PSFTBODefinitionASI">
  <sequence>
   <element name="hostname" type="string"/>
   <element name="ipaddress" type="string"/>
  </sequence>
 </complexType>
</schema>
```

Annotating a business object definition:

You can use the syntax supported by the business object framework to annotate a business object definition.

About this task

The business object framework does not attempt to define the mechanism by which instance metadata data is attached to the business object definition. However, it does define the syntax of the instance metadata.

To annotate the business object definition follow these guidelines.

Procedure

1. If the part of the business object definition that the instance metadata is to be attached to does not already have an annotation, add one. If it already has an annotation, utilize the existing one.
2. Create a separate `xs:appinfo` tag within the `xs:annotation` tag, and define the source attribute to the namespace defined by the business object type metadata being added.
3. Inside the `xs:appinfo` tag, define a QName, using a target namespace prefix, followed by the name of the complex type definition. Append the attribute QName defined with `xmlns:` followed by the previously used target namespace prefix. Set the value of this QName attribute to the target namespace of the business object type metadata being used.
4. Add the structure and instance data mandated by the business object type metadata definition. Close all tags as appropriate.

Example

Once the object is imported in, it is identified to need a set of PeopleSoft annotations. This example shows the same business object definition with the PeopleSoft metadata added.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:p="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified

 <complexType name="Product">
  <annotation>
   <appinfo source="http://www.ibm.com/xmlns/prod/websphere/Adapter/PSFT">
    <psft:PSFTMetadata
     xmlns:psft="http://www.ibm.com/xmlns/prod/websphere/Adapter/PSFT">
```

```
     <hostname>mumbai</hostname>
     <ipaddress>9.29.1.1</ipaddress>
    <psft:PSFTMetadata>
   </appInfo>
   <appInfo>
    <SCMEditor value="Bottom" type="Anchor"/>
   </appInfo>
   <documentation>
    Describes the SCM Product
   </documentation>
  </annotation>

  <sequence>
   <element name="id" type="ID"/>
   <element name="description" type="string" default="DefaultDescription"/>
   <element name="sku" type="p:Sku"/>
  </sequence>
 </complexType>

 <simpleType name="Sku">
  <restriction base="string">
   <pattern value="\d{3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>
</schema>
```

Converting annotation into DataObjects:

The business object framework provides the capability to transform annotations into a usable DataObject structure.

About this task

The annotations associated with a business object definition can be read at runtime by using the SDO implementation-specific set of APIs. However, the problem with these APIs is that they return a binary large object (BLOB). However, the business object framework provides a utility that, if the recommended annotation patterns have been followed, reads the BLOB, validates it, and transforms it into a usable DataObject structure.

To convert an annotation into a DataObject.

Procedure

1. Obtain an annotation.
2. Use the BOTypeMetadata to convert that annotation into an SDO DataObject. The BOTypeMetadata implementation is available as a singleton using the BOTypeMetadata.INSTANCE instance.

Example

The following example demonstrates how to use the APIs to obtain an annotation and then the use the BOTypeMetadata API to convert that annotation into an SDO DataObject. The metadata is defined in `BOTypeMetadata.xsd`.

```
<schema>
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
```

```
 elementFormDefault="qualified">

 <complexType name="VerbInfo">
  <sequence>
   <element name="verbInfo" type="string"/>
  </sequence>
 </complexType>
</schema>
```

One of the capabilities of the business object framework is the ability to add additional supported verbs and accompanying metadata to enable meta-driven capabilities at runtime. The capability is supported through the use of verbs and the VerbInfo metadata to annotate the verbs with additional metadata. The following example demonstrates the places where the VerbInfo metadata would be added for each of the possible Verb values.

```
<schema
 targetNamespace="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pcbg="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:sdo="commonj.sdo"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="http://www.ibm.com/xmlns/prod/websphere/bo/7.0.0"
  schemaLocation="BusinessGraph.xsd"/>

 <import namespace="commonj.sdo"
  schemaLocation="DataGraph.xsd"/>

 <import namespace="http://www.scm.com/ProductCategoryTypes"
  schemaLocation="ProductCategoryTypes.xsd"/>

 <complexType name="ProductCategoryBG">
  <complexContent>
   <extension base="bo:BusinessGraph">
    <sequence>

     <element name="verb"
      minOccurs="0" maxOccurs="1">
      <simpleType>
       <restriction base="string">
        <enumeration value="Create">
         <annotation>
          <appinfo source="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
           <botm:VerbInfo
            xmlns:botm="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
            <verbInfo>Metadata relating to Create</verbInfo>
           </botm:VerbInfo>
          </appinfo>
         </annotation>
        </enumeration>
        <enumeration value="Retrieve">
         <annotation>
          <appinfo source="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
           <botm:VerbInfo
            xmlns:botm="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
            <verbInfo>Metadata relating to Retrieve</verbInfo>
           </botm:VerbInfo>
          </appinfo>
```

```
        </annotation>
       </enumeration>
      </restriction>
     </simpleType>
    </element>

    <element name="productCategory" type="pc:ProductCategory"
     minOccurs="0" maxOccurs="1"/>

   </sequence>
  </extension>
 </complexContent>
</complexType>

<element name="productCategoryBG" type="pcbg:ProductCategoryBG"/>

</schema>
```

Programming using business object services

To facilitate the creation and manipulation of business objects, the business object framework extends SDO specifications by providing a set of Java services. These services are part of the package named `com.ibm.websphere.bo`.

This is a brief description of the business object services.

Table 65. Business object services

Service	Description
BOChangeSummary	Provides enhancements to the SDO change summary interface to manage the business graph's change summary header.
BOCopy	Facilitates copying a graph of business objects or a business graph that contains a graph of business objects.
BODataObject	Provides enhancements to the SDO Data Object interface.
BOEquality	Provides the ability to determine if two business graphs or business objects are equivalent.
BOEventSummary	Provides the interface for managing the content of the business graph's event summary header.
BOFactory	Provides the capability to create a business graph or a business object.
BOMode	Provides access to the business object parsing mode configuration while the application is running. Use this API in the design for applications with logic that provides specific behavior for lazy or eager parsing mode.
BOPrinter	Prints the business object content or type information.
BOType	Provides a mechanism to obtain the SDO type of a business graph or business object that mirrors what `Class.forName()` provides for Java class names.
BOTypeMetadata	Provides the capability of taking an annotation binary large object (BLOB) that conforms to the BO Type Metadata pattern and transforms it into a set of SDO DataObjects (and performs the reserve transform).
BOXMLDocument/BOXMLSerializer	Provides the mechanisms for creating and representing an XML Document in memory, and for serializing and de-serializing an XML document.

Table 65. Business object services (continued)

Service	Description
BOInstanceValidator	Provides a facility to validate a business object instance against its XSD definition. Business objects can be present with various forms. They can be simple business objects or wrapped by the enriched business graph model. In certain business integration scenarios, the business objects are in the deleted section of change summary. These business objects drive the downstream business logics. The accuracy of the business objects need to be ensured in all cases. There are two supported styles for BOInstanceValidator: • Explicit Programmatic Validation: A system service is provided to validate business objects via a set of programming APIs. • Implicit Interface Validation: This validation is enabled/disabled via IBM Integration Designer on the target interfaces via a SCA interface qualifier.

XML document validation

XML documents and business objects can be validated using the validation service.

In addition, other services require certain minimum standards or they throw a runtime exception. One of these is `BOXMLSerializer`.

You can use the `BOXMLSerializer` to validate XML documents before they are processed by a service request. The `BOXMLSerializer` validates the structure of XML documents to determine if any of the following types of errors are present:

- Invalid XML documents, such as those that are missing certain element tags.
- Not well-formed XML documents, such as those that contain missing closing tags.
- Documents containing parsing errors, such as errors in entity declaration.

When an error is discovered by the `BOXMLSerializer`, an exception will be thrown with problem details.

The validation can be performed for import and/or export of XML documents for the following services:

- HTTP
- JAXRPC web services
- JAX-WS web services
- JMS services
- MQ services

For the HTTP, JAXRPC, and JAX-WS services, the `BOXMLSerializer` will generate exceptions in the following manner:

- Imports –
  1. The SCA component invokes the service.
  2. The service invokes a destination URL.
  3. The destination URL responds with an invalid XML exception.
  4. The service fails with a runtime exception and message.
- Exports –

1. The service client invokes the service export.
2. The service client sends an invalid XML
3. The export fails for the service and generates an exception and message.

For the JMS and MQ messaging services, the exceptions are generated in the following manner:

- Imports –
  1. The import invokes the JMS or MQ service.
  2. The service returns a response.
  3. The service returns an invalid XML exception.
  4. The import fails and generates a message.
- Exports –
  1. The MQ or JMS client invokes an export.
  2. The client sends invalid XML.
  3. The export fails and generates an exception and message.

You can view the logs for any messages generated by an XML validation exception. The following examples are messages generated by improper XML coding that was validated by the `BOXMLSerializer`

- JAXWS import

```
javax.xml.ws.WebServiceException: org.apache.axiom.om.OMException:
 javax.xml.stream.XMLStreamException: Element type "TestResponse" must be
 followed by either attribute specifications, ">" or "/>".

javax.xml.ws.WebServiceException: org.apache.axiom.soap.SOAPProcessingException:
 First Element must contain the local name, Envelope
```

- JAXRPC import

```
[9/11/08 15:16:27:417 CDT] 0000003e ExceptionUtil E
 CNTR0020E: EJB threw an unexpected (non-declared)
  exception during invocation of method
  "transactionNotSupportedActivitySessionNotSupported" on bean
  "BeanId(WSXMLValidationApp#WSXMLValidationEJB.jar#Module, null)".
  Exception data: WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: org.xml.sax.SAXParseException: Element type "TestResponse"
   must be followed by either
  attribute specifications, ">" or "/>". Message being parsed:
    <?xml version="1.0"?><TestResponse
  xmlns="http://WSXMLValidation"<firstName>Bob</firstName>
    <lastName>Smith</lastName></TestResponse>
 faultActor: null
 faultDetail:
[9/11/08 15:16:35:135 CDT] 0000003f ExceptionUtil E   CNTR0020E: EJB threw an
  unexpected (non-declared) exception during invocation of method
  "transactionNotSupportedActivitySessionNotSupported" on bean
  "BeanId(WSXMLValidationApp#WSXMLValidationEJB.jar#Module, null)".
 Exception data: WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: org.xml.sax.SAXException: WSWS3066E: Error: Expected 'envelope'
   but found  TestResponse
  Message being parsed: <?xml version="1.0"?><TestResponse
   xmlns="http://WSXMLValidation">
  <firstName>Bob</firstName><middleName>John</middleName>
   <lastName>Smith</lastName>
   </TestResponse>
 faultActor: null
 faultDetail:
```

- JAXRPC/JAXWS export

```
[9/11/08 15:35:13:401 CDT] 00000064 WebServicesSe E
  com.ibm.ws.webservices.engine.transport.http.WebServicesServlet
  getSoapAction WSWS3112E:
   Error: Generating WebServicesFault due to missing SOAPAction.
                    WebServicesFault
 faultCode: Client.NoSOAPAction
 faultString: WSWS3147E: Error: no SOAPAction header!
 faultActor: null
 faultDetail:
```

For more information about validation services, see the `BOInstanceValidator` interface in the Generated API and SPI documentation in the Reference section.

Programming techniques

These techniques illustrate how to effectively program business objects using the business object framework.

Arrays in business objects

You can define arrays for an element in a business object so that the element can contain more than one instance of data.

You can use a List type to create an array for a single named element in a business object. This will allow you to use that element to contain multiple instances of data. For example, you can use an array to store several telephone numbers within an element named `telephone` that is defined as a string in its business object wrapper. You can also define the size of the array by specifying the number of data instances using the `maxOccurs` value. The following example code shows how you would create such an array that will hold three instances of data for that element:

```
<xsd:element name="telephone" type="xsd:string" maxOccurs="3"/>
```

This will create a list index for the element `telephone` that can hold up to three data instances. You may also use the value `minOccurs` if you are only planning to have one item in the array.

The resulting array consists of two items:

- the contents of the array
- the array itself.

In order to create this array, however, you need perform an intermediate step by defining a wrapper. This wrapper, in effect, replaces the property of the element with an array object. In the previous example, you can create an `ArrayOfTelephone` object to define the element `telephone` as an array. The following code example shows how you accomplish this task:

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="Customer">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="ArrayOfTelephone" type="ArrayOfTelephone"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="ArrayOfTelephone">
```

```
      <xsd:sequence maxOccurs="3">
        <xsd:element name="telephone" type="xsd:string" nillable="true"/>
      </xsd:sequence>
    </xsd:complexType>
 </xsd:schema>
```

The `telephone` element now appears as a child of the `ArrayOfTelephone` wrapper object.

Note that in the previous example, the `telephone` element contains a property named `nillable`. You can set this property to `true` if you want to certain items in the array index to contain no data. The following example code shows how the data in an array may be represented:

```
<Customer>
   <name>Bob</name>
   <ArrayOfTelephone>
     <telephone>111-1111</telephone>
     <telephone xsi:nil="true"/>
     <telephone>333-3333</telephone>
   </ArrayOfTelephone>
 </Customer>
```

In this case, the first and third items in the array index for the `telephone` element contain data, while the second item does not contain any data. If you had not used the `nillable` property for the `telephone` element, then you would have had to have the first two elements contain data.

You can use the Service Data Object (SDO) Sequence APIs in WebSphere ESB as an alternative method to handle sequences in business object arrays. The following example code creates an array for the `telephone` element with data identical to that shown in the previous example:

```
DataObject customer = ...
 customer.setString("name", "Bob");

 DataObject tele_array = customer.createDataObject("ArrayOfTelephone");
 Sequence seq = tele_array.getSequence();  // The array is sequenced
 seq.add("telephone", "111-1111");
 seq.add("telephone", null);
 seq.add("telephone", "333-3333");
```

You can return the data for a given element array index by using code similar to the example below:

```
String tele3 = tele_array.get("telephone[3]");  // tele3 = "333-3333"
```

In this example, a string named `tele3` will return the data "333-3333".

You can enter the data items for the array in the list index by using fixed width or delimited data placed in a JMS or MQ message queue. You can also accomplish this task by using a flat text file that contains the properly formatted data

Creating nested business objects

You can use the `setWithCreate` function to create nested business objects within a parent business object.

You can create nested business objects from a parent business object without having to write code that details intermediate child objects. For instance, you can set a nested business object two levels below the parent object without having to

define a dependent business object one level below the parent object. Use the `setWithCreate` function to accomplish this task for:

- a single instance
- multiple instances
- a wildcard value
- a model group

The following topics describe how you can do each of these.

Single instance of a nested business object:

Use the `setWithCreate` function to create a single instance of nested business object.

Before you begin

The following example code shows how you would normally have to create code for an intermediate (child) object from a higher level (parent) object in order to create a third-level (grandchild) object. The XSD file would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

About this task

If you used the traditional "top-down" method to set the business object data, you would have to process the following code specifying the child and grandchild objects before setting the data in the grandchild object:

```
DataObject parent = ...
DataObject child = parent.createDataObject("child");
DataObject grandchild = child.createDataObject("grandChild");
grandchild.setString("name", "Bob");
```

You can use a more efficient method by using the `setWithCreate` function to simultaneously define the grandchild object and set its data, without having to specify the intermediate child object. The following example code shows how you would accomplish this task:

```
DataObject parent = ...
parent.setString("child/grandchild/name", "Bob");
```

Results

The lower-level business object data is set without having to reference the intermediate-level business object. An exception occurs if the path is not valid.

Creating multiple instances of nested business objects:

Use the `setWithCreate` function to create a multiple instances of nested business object.

Before you begin

The following example XSD file contains nested objects one (child) and two (grandchild) levels below the top (parent) business object:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child" maxOccurs="5"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Note that the parent object can have up to five child objects, as specified in the `maxOccurs` value.

About this task

You can create a list with a more stringent policy that will not allow for missing sequences in an array. You can use the `setWithGet` method, and at the same time specify the data that will appear in a particular list index item:

```
DataObject parent = ...
parent.setString("child[3]/grandchild/name", "Bob");
```

In this case, the resulting array would be of size three, but the values for `child[1]` and `child[2]` list index items are undefined. You may want the items to either be a null value or have an associated data value. In the scenario above, an exception will be thrown because the values for the first two array index items are undefined.

You can remedy this situation by defining the values in the index of the list. If the index item refers to an existing element in the array and if that element is not null (that is, it contains data), it will be used. If it is null, it will be created and used. If the index of the list is one greater than the size of the list, a new value will be created and added. The following example code shows what will happen in a list that of size two, where `child[1]` is designated null and `child[2]` contains data:

```
DataObject parent = ...
// child[1] = null
// child[2] = existing Child
// This code will work because child[1] is null and will be created.
parent.setString("child[1]/grandchild/name", "Bob");

// This code will work because child[2] exists and will be used.
parent.setString("child[2]/grandchild/name", "Dan");

// This code will work because the child list is of size 2, and adding
// one more list item will increase the list size.
parent.setString("child[3]/grandchild/name", "Sam");
```

Results

You have overridden the values for the two existing items and added a third item to the list index. If, however, you then add another item that is not of size four, or which is greater than the size specified in `maxOccurs`, then an exception will be thrown. The more stringent policy of this method is demonstrated in the following example code.

Note: The following code is assumed to be appended to the existing code shown in the "About this task" section:

```
// This code will throw an exception because the list is of size 3
// and you have not created an item to increase the size to 4.
parent.setString("child[5]/grandchild/name", "Billy");
```

Using a nested business object defined by a wildcard:

You can specify the type `xsd:any` in a parent object to specify a child object, but only if the child object already exists.

About this task

The `setWithCreate` function used to define nested business objects for single and multiple instances does not work if you are using a wildcard value of `xsd:any` in the Service Data Object. This is illustrated in the following example code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="xsd:anyType"/>
```

```
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Results

An exception will be thrown if the child data object does not exist.

Using business objects in model groups:

You create the model group path patterns when working with nested business objects that are part of a model group.

About this task

Model groups use the tag `xsd:choice` that you can use to create business objects from a parent business object. The business object framework, however, can cause naming conflicts that can generate an exception. The following example code illustrates how naming conflicts can occur:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Note: there can be multiple instances of the elements named "child1" and "child2",

Use the Service Data Object (SDO) path patterns for model groups to resolve these conflicts.

Results

You would get arrays that use the SDO path pattern that is used to handle model groups, as shown in the following example code:

```
set("child1/grandchild/name", "Bob");

set("child11/grandchild/name", "Joe");
```

Differentiating identically named elements

You must provide unique names for business object elements and attributes.

In the Service Data Object (SDO) framework, elements and attributes are created as properties. In the following code examples, the XSDs create types that have one property named foo:

```
<xsd:complexType name="ElementFoo">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AttributeFoo">
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>
```

In these cases, you can access the property using the XML Path Language (XPath). However, valid schema types can have an attribute and element of the same name, as in the following example:

```
<xsd:complexType name="DuplicateNames">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>
```

In XPath, you must be able to differentiate identically named elements from attributes. This is achieved by beginning one of the names with an at sign (@). The following snippet shows how to access the identically named element and attribute:

```
1 DataObject duplicateNames = ...

2 // Displays "elem_value"
3 System.out.println(duplicateNames.get("foo"));

4 // Displays "attr_value"
5 System.out.println(duplicateNames.get("@foo"));
```

Use this naming scheme for all methods that take a String value that is an SDO XPath.

Model group support (all, choice, sequence, and group references):

The SDO specification requires model groups (all, choice, sequence, and group references) to be expanded in place and not describe types or properties.

Basically, this means that any of those structures that are within the same containing structures are "flattened". This "flattening" puts all the child structures at the same level, which can produce duplicate naming issues in an SDO whose structure is derived from the flattened data. When an XSD does not flatten the groups, there is still a separation of duplicates that are contained by different parents.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
```

```
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Since the multiple occurrences of option1 and option2 are contained in different choice blocks and even have a separating element between them, XSD and XML have no problem distinguishing between them. But when SDO flattens these groups, all the option properties are now under the same container of MultipleGroup.

Even without duplicate names, there is also the semantic issue that the flattening of these groups cause. Take the following XSD for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://SimpleChoice">
  <xsd:complexType name="SimpleChoice">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Asking the user to rename duplicate names or add special annotations to XSDs is impractical because in many cases, like standards and industry schemas, the user does not control the XSDs they are working with.

To create consistency for all properties, business objects include a method to access each individual occurrence of the duplicate named properties through XPath. Following the business object framework naming convention, any duplicate property names encountered have the next unused digit appended to their name So for example, the following XSD:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://TieredGroup">
  <xsd:complexType name="TieredGroup">
    <xsd:sequence>
      <xsd:choice minOccurs="0">
        <xsd:sequence>
          <xsd:element name="low" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:choice minOccurs="0">
            <xsd:element name="width" minOccurs="0"
      maxOccurs="1" type="xsd:string"/>
            <xsd:element name="high" minOccurs="0"
      maxOccurs="1" type="xsd:string"/>
          </xsd:choice>
        </xsd:sequence>
        <xsd:element name="high" minOccurs="1"
    maxOccurs="1" type="xsd:string"/>
        <xsd:sequence>
          <xsd:element name="width" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
```

```
          <xsd:element name="high" minOccurs="0"
     maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element name="center" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:element name="width" minOccurs="0"
     maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

The preceding XSD produces the following DataObject model:

```
DataObject - TieredGroup
 Property[0] - low - string
 Property[1] - width - string
 Property[2] - high - string
 Property[3] - high1 - string
 Property[4] - width1 - string
 Property[5] - high2 - string
 Property[6] - center - string
 Property[7] - width2 - string
```

Where **width**, **width1**, and **width2** are the names of the properties named width starting from the first one in the XSD going down, likewise with **high**, **high1**, **high2**.

These new property names are just the names used for reference and XPath and do not affect serialized content. The "true" names of each of these properties that appear in the serialized XML are the values given in the XSD. So for the XML instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:TieredGroup xsi:type="p:TieredGroup"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://TieredGroup">
  <width>foo</width>
  <high>bar</high>
</p:TieredGroup>
```

In order to access those properties you would use the following code:

```
 DataObject tieredGroup = ...

 // Displays "foo"
 System.out.println(tieredGroup.get("width1"));

 // Displays "bar"
 System.out.println(tieredGroup.get("high2"));
```

Differentiating identically named properties

When multiple XSDs with the same namespace define the same named types, an incorrect type can be accidentally referenced.

Address1.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
```

```
    <xsd:sequence>
      <xsd:element minOccurs="0" name="city" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Address2.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="state" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Business objects do not support duplicate names for any global XSD structures (such as complexType, simpleType, element, attribute, and so on) through the BOFactory.create() APIs. These duplicate global structures can still be created as the child to other structures if the proper APIs are used, as shown in the following examples

Customer1.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Customer1"
 targetNamespace="http://Customer1">
  <xsd:import schemaLocation="./Address1.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Customer2.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Customer2"
 targetNamespace="http://Customer2">
  <xsd:import schemaLocation="./Address2.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

When populating both of the Customer address fields and then calling BOFactory.create() to make the Address, the resulting child business object types can be incorrectly set. You can avoid this by calling the createDataObject(“address”) API on the Customer DataObject. This will be guaranteed to produce a child of the correct type because business objects will follow the import's schemaLocation.

```
 DataObject customer1 = ...

 // Incorrect way to create Address child
 // This may create a type of Address1.xsd Address or maybe Address2.xsd Address
```

```
DataObject incorrect = boFactory.create("", "Address");
customer1.set("address", incorrect);

// Correct way to create Address child
// This is guaranteed to create a type of Address1.xsd Address
customer1.createDataObject("address");
```

Resolving property names that contain periods

Property names in an XSD may contain a period (".") as one of many valid characters, while in a SDO they are also used to show indexing in a property of multiple cardinality. This may cause resolution problems in certain situations.

Property names in Service Data Objects (SDOs) are based on the names of the elements and attribute that are generated from in the XSD. Business objects will handle the "." character properly, with one exception: if an XSD has a single cardinality property named "<name>.<#>" and a multiple cardinality property named "<name>".

An XPath such as "foo.0" would not resolve properly if there is a single cardinality property named "foo.0" and multiple cardinality property named "foo". In this case, the single cardinality Property named "foo.0" would be the one resolved. Although this should be a rare occurrence, you can avoid it entirely if you use the "foo[1]" syntax to access their multiple cardinality property. SDOs will not support the "." syntax for indexing, so you should use the "[]" for indexing.

Serializing and deserializing unions with xsi:type:

In XSD, a union is a way to merge the lexical spaces of several simple datatypes known as members.

The following example XSD shows a union that has the members of an integer and a date.

```
<xsd:simpleType name="integerOrDate">
 <xsd:union memberTypes="xsd:integer xsd:date"/>
</xsd:simpleType>
```

This multiple typing can cause confusion during deserialization and when manipulating the data.

Business objects support SDO's using xsi:type for serialization and will follow the same algorithm for determining the type on a deserialization if the xsi:type is not present in the XML data.

So to guarantee that the data (the number "42" in this example) would be deserialized as an integer, you can use the xsi:type specified in the input XML. You can also order the member list of the union in the XSD so that the integer comes before the string. The following example shows how both methods are implemented:

```
<integerOrString xsi:type="xsd:integer">42</integerOrString>

<xsd:simpleType name="integerOrString">
 <xsd:union memberTypes="xsd:integer xsd:string"/>
</xsd:simpleType>
```

Likewise, if the user wanted the data to be deserialized as a string, then either of the following changes would cause that behavior:

```
<integerOrString xsi:type="xsd:string">42</integerOrString>

<xsd:simpleType name="integerOrString">
 <xsd:union memberTypes="xsd:string xsd:integer"/>
</xsd:simpleType>
```

Note that if a string type is the first member of the union, it never has any information loss. It can also hold any data that will always be chosen by the no xsi:type algorithm. If you want to use a type other than string, you must either use xsi:type in the XML or reorder the member types in the XSD to give the other members a chance to accept the data.

Support for null business objects

This scenario involves an outside system communicating with WebSphere ESB through XML wrapped inside of a SOAP message. When the enclosed element is nillible and has `xsi:nil="true"`, then the resulting DataObject which is created in WebSphere ESB is null.

Here is an example which illustrates an XML message with a nillible element.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
 <soap:Body>
  <p:Employee xmlns:p="http://www.mycompany.com" xmlns:xsi="http://www.w3.org"
   xsi:nil="true"/>
 </soap:Body>
</soap:Envelope>
```

Where `Employee` is defined as:

```
<element name="Employee" nillable="true">
...
</element>
```

The business object that is generated and sent from the export is null in this case. For example, if any downstream components have operations invoked on them, the input to that operation is null.

Note: These objects cannot be passed into business object maps because business object maps are unable to map fields from a null object.

Using the Sequence object to set data order

Some XSDs are defined in a way that makes the order that the data occurs in the XML have special significance.

One example of order significance in XSDs is mixed content. If the text data appears before or after an element, it may have different meaning than if it occurs in a different location. For these situations, SDO generates an object known as a Sequence, which is used to set the data in an ordered fashion.

SDO Sequences should not be confused with XSD sequences. XSD sequences are just model groups that are flattened out before SDO model generation. The presence of an XSD sequence does not relate to the presence of an SDO Sequence.

The following conditions in an XSD cause an SDO Sequence to be generated:

A complexType with mixed content:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://MixedContent"
 targetNamespace="http://MixedContent">
  <xsd:complexType name="MixedContent" mixed="true">
    <xsd:sequence>
      <xsd:element name="element1" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element2" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element3" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="MixedContent" type="tns:MixedContent"/>
</xsd:schema>
```

A schema that has 1 or more <any/> tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyElemAny"
 targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <xsd:any/>
      <xsd:element name="marker1" type="xsd:string"/>
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

A model group array (an all, choice, sequence, or group reference with maxOccurs > 1):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence maxOccurs="3">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

An <all/> tag of maxOccurs <= 1 that contains more than one element:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://All">
  <xsd:complexType name="All">
    <xsd:all>
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:schema>
```

Specific information about using <any/> and sequence together will be discussed in the topic listed under “Related Information” on this page. The general information that follows in the remainder of this section will describe how to work with the other Sequence conditions, but will still apply to <any/> as well.

How do I know if my DataObject has a sequence?:

There are two simple APIs to choose from that can determine if a DataObject is sequenced: DataObject noSequence and DataObject withSequence.

You would use DataObject noSequence and DataObject withSequence as shown in the following example:

```
DataObject noSequence = ...
DataObject withSequence = ...

// Displays false
System.out.println(noSequence.getType().isSequenced());

// Displays true
System.out.println(withSequence.getType().isSequenced());

// Displays true
System.out.println(noSequence.getSequence() == null);

// Displays false
System.out.println(withSequence.getSequence() == null);
```

Why do I need to know a DataObject has a Sequence?:

If you are working on a DataObject that has a Sequence, it is important to know the order in which the data is set. Because of this, care must be taken in the order in which the values are set.

A DataObject that is not sequenced allows random order set access. This functions like a Map where all the keys are set to the same values. It does not matter in what order the keys were set, the data in the map is the same, and would be serialized to XML identically.

When a DataObject is sequenced, the order in which the data was set is recorded in the Sequence, much like adding data to a List. This provides two ways to access the data, by name/value pairs (the DataObject APIs) and by order in which it was set (the Sequence APIs). You can use the DataObject set(...) or Sequence add(...) APIs to preserve the structure. This ordering affects the way that the XML is serialized.

Take for example, the <all/> tag XSD. When the set methods are called in the following order it produces the following XML when serialized:

```
DataObject all = ...
all.set("element1", "foo");
all.set("element2", "bar");

<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:p="http://All">
  <element1>foo</element1>
  <element2>bar</element2>
</p:All>
```

If instead, the set methods are called in the opposite order, then the following XML is produced when the business object is serialized:

```
 DataObject all = ...
 all.set("element2", "bar");
 all.set("element1", "foo");
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:p="http://All">
  <element2>bar</element2>
  <element1>foo</element1>
</p:All>
```

If the order of the Sequence is ever changed, then the Sequence class has basic add, remove, and move methods to allow the user to alter the order of Sequence.

How do I work with mixed content?:

For mixed content, Sequence has a specific API for adding text: addText(...).

All other APIs work equally with text as they do with Properties. The getProperty(int) API will return null for mixed content text data. The following example of mixed content code can be used to print all the mixed content text from a DataObject:

```
 DataObject mixedContent = ...
 Sequence seq = mixedContent.getSequence();

 for (int i=0; i < seq.size(); i++)
 {
     Property prop = seq.getProperty(i);
     Object value = seq.getValue(i);

     if (prop == null)
     {
         System.out.println("Found mixed content text: "+value);
     }
     else
     {
         System.out.println("Found Property "+prop.getName()+": "+value);
     }
 }
```

How do I work with a model group array?:

A model group array is created when a model group has a value for maxOccurs > 1.

Since model groups are flattened and not expressed in a DataObject, the properties inside of the model group become multiple cardinality properties so that their isMany() methods return true if they are not already true. Their minOccurs and maxOccurs facets become multiplied by that of the containing model group. Choice will multiply the maxOccurs facet in the same way as the other model groups, but will always use 0 as the multiplication value for minOccurs because any data in a choice may not be selected.

For example, the following XSD has a model group array:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
```

```
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
   minOccurs="0" maxOccurs="3"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

As stated, **element1** and **element2** will now be multiple cardinality so that a get(...) accessor would return a List. **Element1** has a default minOccurs of 1 and a default maxOccurs of 1. **Element2** has a minOccurs of 0 and a maxOccurs of 3. In the following example, their new minOccurs and maxOccurs will be as follows:

```
DataObject - ModelGroupArray
 Property[0] - element1 - minOccurs=(2*1)=2 - maxOccurs=(5*1)=5
 Property[1] - element2 - minOccurs=(2*0)=0 - maxOccurs=(5*3)=15
```

If the type were Choice:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:choice minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
   minOccurs="0" maxOccurs="3"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Would generate the following minOccurs due to the exclusion of choice that only **element1** can be picked each time or only **element2** could be picked each time, so to pass validation both need to be able to have 0 occurrences:

```
DataObject - ModelGroupArray
 Property[0] - element1 - minOccurs=(0*1)=0 - maxOccurs=(5*1)=5
 Property[1] - element2 - minOccurs=(0*0)=0 - maxOccurs=(5*3)=15
```

Using Any data types

This section provides programming techniques for using Any data types.

Using AnySimpleType for simple types:

AnySimpleType is handled by the business object API in a manner that is similar to other simple types (such as string, int, boolean, and so on).

The only differences between anySimpleType and the other simple types are in which instance data can be mapped to or from the field and how this instance data is serialized.

If a string type were to have a set(...) method called on it, the data would first be converted to a string, and the original data type would be lost:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://StringType">
  <xsd:complexType name="StringType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:string"/>
```

```
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject stringType = ...

 // Set the data to a String
 stringType.set("foo", "bar");

 // The instance data will always be type String, regardless of the data set
 // Displays "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());

 // Set the data to an Integer
 stringType.set("foo", new Integer(42));

 // The instance data will always be type String, regardless of the data set
 // Displays "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());
```

If anySimpleType were to have a set(...) method called on it, the original data would not be lost.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://AnySimpleType">
  <xsd:complexType name="AnySimpleType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:anySimpleType"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject anySimpleType = ...

 // Set the data to a String
 anySimpleType.set("foo", "bar");

 // The instance data will always be of the type of data used in the set
 // Displays "java.lang.String"
 System.out.println(anySimpleType.get("foo").getClass().getName());

 // Set the data to an Integer
 anySimpleType.set("foo", new Integer(42));

 // The instance data will always be of the type of data used in the set
 // Displays "java.lang.Integer"
 System.out.println(anySimpleType.get("foo").getClass().getName());
```

This data type is also preserved across serialization and deserialization by xsi:type. Consequently, any time you serialize an anySimpleType element, it will have an xsi:type that matches that defined in the SDO specification based on its Java type:

In the following example, you serialize the business object so that the data would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:StringType xsi:type="p:StringType"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
```

```
 xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:p="http://StringType">
  <foo xsi:type="xsd:int">42</foo>
<p:StringType></p:StringType>
```

The xsi:type will be used during deserialization to load the data as the appropriate Java instance class. If no xsi:type is specified, the default deserialization type will be string.

For the other simple types, determining how the data can be mapped is a constant. For instance, a boolean can always map to a string. AnySimpleType can contain any of the simple types, however, so whether a mapping is possible is based on the instance data in the field.

Use the property Type URI and Name to determine if a property is of type anySimpleType. They will be “commonj.sdo” and “Object”. To determine if data is valid to be inserted into anySimpleType, check to see if it is not an instance of a DataObject. Data that can be represented as a string and that is not a DataObject can be set in an anySimpleType field.

This leads to the following mapping rules:

- anySimpleType can always map to anySimpleType.
- any other simple type can always map to anySimpleType.
- anySimpleType can always map to string because all simple types must be able to be converted into a string.
- anySimpleType might be able to map to any of the other simple types, depending on its value in the business object. The mapping cannot be determined at design time; it is determined at run time.

Related information:

Assigning from and to xs:any

Using AnyType for complex types:

The anyType tag is handled by the business object API in a manner that is similar to other complex types.

The only differences between anyType and other complex types are in which instance data can be mapped to or from the field and how this instance data is serialized. Complex types are limited to a single type: Customer, Address, and so on. The anyType, however, allows any DataObject regardless of type. If maxOccurs > 1, then each DataObject in the list can of a different type.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://AnyType">
  <xsd:complexType name="AnyType">
    <xsd:sequence>
      <xsd:element name="person" type="xsd:anyType"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://Customer">
  <xsd:complexType name="Customer">
```

```
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Employee" targetNamespace="http://Employee">
  <xsd:complexType name="Employee">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject anyType = ...
 DataObject customer = ...
 DataObject employee = ...

 // Set the person to a Customer
 anyType.set("person", customer);

 // The instance data will be a Customer
 // Displays "Customer"
 System.out.println(anyType.getDataObject("person").getName());

 // Set the person to an Employee
 anyType.set("person", employee);

 // The instance data will be an Employee
 // Displays "Employee"
 System.out.println(anyType.getDataObject("person").getName());
```

Just like anySimpleType, anyType uses xsi:type during serialization to assure that the intended type of DataObject is maintained when deserialized. When you set to "Customer," the XML would look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:customer="http://Customer"
    xmlns:p="http://AnyType">
  <person xsi:type="customer:Customer">
    <name>foo</name>
  </person>
</p:AnyType>
```

And when set to "Employee":

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:employee="http://Employee"
    xmlns:p="http://AnyType">
  <person xsi:type="employee:Employee">
    <id>foo</id>
  </person>
</p:AnyType>
```

AnyType also allows for the setting of simple type values through wrapper DataObjects. These wrapper DataObjects have a single property named "value" (element) that holds the simple type value. The business object API have been overridden to automatically wrap and unwrap these simple types and wrapper DataObjects when using the get<Type>/set<Type> APIs. The non-type casting get/set APIs will not perform this wrapping.

```
DataObject anyType = ...

// Calling a set<Type> API on an anyType Property causes automatic
// creation of a wrapper DataObject
anyType.setString("person", "foo");

// The regular get/set APIs are not overridden, so they will return
// the wrapper DataObject
DataObject wrapped = anyType.get("person");

// The wrapped DataObject will have the "value" Property
// Displays "foo"
System.out.println(wrapped.getString("value"));

// The get<Type> API will automatically unwrap the DataObject
// Displays "foo"
System.out.println(anyType.getString("person"));
```

When the wrapper DataObject is serialized, it will be serialized just like anySimpleType mapping of Java instance classes to XSD types in the xsi:type field. So this setting would serialize as:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd=" http://www.w3.org/2001/XMLSchema"
 xmlns:p="http://AnyType">
  <person xsi:type="xsd:string">foo</person>
</p:AnyType>
```

If no xsi:type is given or if an incorrect xsi:type is given, then an exception will be thrown. In addition to automatic wrapping, the wrapper can be manually created for use with the set() API through BOFactory createDataTypeWrapper(Type, Object), where Type is the SDO simple type of the data to be wrapped and Object is the data to be wrapped.

```
Type stringType = boType.getType("http://www.w3.org/2001/XMLSchema", "string");
DataObject stringType = boFactory.createByMessage(stringType, "foo");
```

To determine if a DataObject is a wrapper type, the BOType isDataTypeWrapper(Type) can be called.

```
DataObject stringType = ...
boolean isWrapper = boType.isDataTypeWrapper(stringType.getType());
```

For the other complex types, in order to move data from one field to another, the data must be of the same type. AnyType can contain any complex types, however, so a direct move with no mapping might not be possible, based on the instance data in the field.

You can use the property Type URI and Name to determine if a property is of type anyType. They will be "commonj.sdo" and "DataObject". All data is valid to be inserted into an anyType. This leads to the following mapping rules:

- anyType can always map to anyType.

- any complex type can always map to anyType.
- any simple type can always map to anyType.
- anyType might be able to map to any of the other simple or complex types, depending on its value in the business object instance. The mapping cannot be determined at design time; it is determined at run time.

Using Any to set global elements for complex types:

You can use the <any/> tag to set global elements to a complex type.

An occurrence of the any tag makes the DataObject Type isOpen() method and the isSequenced() method return true. If the value for maxOccurs is > 1 on an any tag, it has no effect on the structure of the DataObject; it is only used as information during validation. Similarly, the occurrence of multiple any tags in a type does not change the structure of the DataObject; they are used only for validating the location of open data that was set.

How do I know if my DataObject has an any tag?:

You can easily determine if instances of a DataObject have any values set within them by checking the instance properties to see if any of the open properties are attributes.

DataObject does not provide a mechanism for determining if a DataObject Type has an any tag. DataObjects only have the concept of "open" that applies to both any and anyAttribute and allows the free additional of any properties. While the presence of an any tag causes a DataObject to have isOpen() = true and isSequenced() = true, it might just have an anyAttribute tag and one of the reasons for being sequenced discussed in the Sequences topics. The following example demonstrates these concepts:

```
DataObject dobj = ...

// Check to see if the type is open, if it isn't then it can't have
// any values set in it.
boolean isOpen = dobj.getType().isOpen();

if (!isOpen) return false;  // Does not have any values set

// Open Properties are added to the Instance Property list, but not
// the Property list, so comparing their sizes can easily determine
// if any open data is set
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// If equal, does not have any open content set
if (instancePropertyCount == definedPropertyCount) return false;

// Check the open content Properties to determine if any are Elements
for (int i=definedPropertyCount; i < instancePropertyCount; i++)
{
    Property prop = (Property)dobj.getInstanceProperties().get(i);
    if (boXsdHelper.isElement(prop))
    {
        return true;  // Found an any value
    }
```

```
 }

 return false;  // Does not have any values set
```

How do I get/set any values?:

Performing a get on data that was set in an any field can be done in the same manner as any other element value if the name is known.

You can perform a get with the XPath `<name>`. If the name is unknown, then the value can be found by checking the instance properties. If there are multiple any tags, or an any tag with `maxOccurs > 1`, then the DataObject sequence will have to be used instead if it is important to determine which any tag the data originated from.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyElemAny"
 targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <!-- Handle all these any one way -->
      <xsd:any maxOccurs="3"/>
      <xsd:element name="marker1" type="xsd:string"/>
      <!-- Handle this any in another -->
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Because the <any/> tag causes the DataObject to be sequenced determining which any value was set can be done by checking the Sequence for the position of the any properties.

You can determine which any tag the instance data for the following XSD belongs to by using the following code:

```
 DataObject anyElemAny = ...
 Seqeuence seq = anyElemAny.getSequence();

 // Until we encounter the marker1 element, all the open data
 // found belongs to the first any tag
 boolean foundMarker1 = false;

 for (int i=0; i<seq.size(); i++)
 {
     Property prop = seq.getProperty(i);

     // Check to see if the property is an open property
     if (prop.isOpenContent())
     {
         if (!foundMarker1)
         {
             // Must be the first any because it occurs
             // before the marker1 element
             System.out.println("Found first any data: "+seq.getValue(i));
         }
         else
         {
             // Must be the second any because it occurs
```

```
            // after the marker1 element
            System.out.println("Found second any data: "+seq.getValue(i));
         }
     }
     else
     {
         // Must be the marker1 element
         System.out.println("Found marker1 data: "+seq.getValue(i));
         foundMarker1 = true;
     }
 }
```

Setting an <any/> value is done by creating a global element property and adding that value to the sequence.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://GlobalElems"
 targetNamespace="http://GlobalElems">
  <xsd:element name="globalElement1" type="xsd:string"/>
  <xsd:element name="globalElement2" type="xsd:string"/>
</xsd:schema>

 DataObject anyElemAny = ...
 Seqeuence seq = anyElemAny.getSequence();

 // Get the global element Property for globalElement1
 Property globalProp1 = boXsdHelper.getGlobalProperty(http://GlobalElems,
 "globalElement1", true);

 // Get the global element Property for globalElement2
 Property globalProp2 = boXsdHelper.getGlobalProperty(http://GlobalElems,
 "globalElement2", true);

 // Add the data to the sequence for the first any
 seq.add(globalProp1, "foo");
 seq.add(globalProp1, "bar");

 // Add the data for the marker1
 seq.add("marker1", "separator");  // or anyElemAny.set("marker1", "separator")

 // Add the data to the sequence for the second any
 seq.add(globalProp2, "baz");

 // The data can now be accessed by a get call
 System.out.println(dobj.get("globalElement1"); // Displays "[foo, bar]"
 System.out.println(dobj.get("marker1");        // Displays "separator"
 System.out.println(dobj.get("globalElement2"); // Displays "baz"
```

Valid mappings for data in an any attribute:

An <any/> tag is a set of name/value pairs. The only valid mapping that can be determined at design time for <any/> is another <any/> or anyType that has the same maxOccurs value.

Individually, the values contained in an instance of a DataObject for any are basic complex types that follow all the rules of complex type mapping. Some of these complex types may be wrapped simple types, so these will follow the rules of simple type mapping.

Using AnyAttribute to set global attributes for complex types:

The <anyAttribute/> tag allows a complex type to have any number of global attributes set to it.

Similar to the <any/> tag, the occurrence of the <anyAttribute/> tag makes the DataObject Type isOpen() method return true. Unlike the <any/> tag, however, the <anyAttribute/> tag does not cause a DataObject to be sequenced because attributes in XSD are not ordered constructs.

How do I tell if my DataObject has an anyAttribute tag?:

You can easily determine if instances of a DataObject have anyAttribute values set within them by checking the instance properties to see if any of the open properties are attributes.

DataObject does not provide a mechanism for determining if a DataObject Type has an anyAttribute tag. DataObjects only have the concept of "open" that applies to both any and <anyAttribute/> and allows the free additional of any Properties. While it is true that if a DataObject has isOpen() = true and isSequenced() = false, then it must have an anyAttribute tag, if isOpen() = true and isSequenced() = true, the DataObject Type might or might not have an anyAttribute tag.

DataObject provides metadata query methods to programmatically answer this and other questions about the XSD structure that was used to generate the DataObject. The InfoSet model can be queried if it is necessary to know if the anyAttribute tag is present. Because anyAttribute is singular and either is or is not true, business objects will also provide a BOXSDHelper hasAnyAttribute(Type) method to allow determination as to if setting an open attribute on this DataObject would produce a valid result. The following example code demonstrates these concepts:

```
DataObject dobj = ...

// Check to see if the type is open, if it isn't then it can't have
// anyAttribute values set in it.
boolean isOpen = dobj.getType().isOpen();

if (!isOpen) return false;  // Does not have anyAttribute values set

// Open Properties are added to the Instance Property list, but not
// the Property list, so comparing their sizes can easily determine
// if any open data is set
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// If equal, does not have any open content set
if (instancePropertyCount == definedPropertyCount) return false;

// Check the open content Properties to determine if any are Attributes
for (int i=definedPropertyCount; i<instancePropertyCount; i++)
{
    Property prop = (Property)dobj.getInstanceProperties().get(i);
    if (boXsdHelper.isAttribute(prop))
    {
        return true;  // Found an anyAttribute value
    }
}

return false;  // Does not have anyAttribute values set
```

How do I get/set anyAttribute values?:

Setting an <anyAttribute/> value is done in the same way as setting an <any/> value, but instead of a global element a global attribute is used.

Performing a get on an attribute value that was set in an anyAttribute field can be done in the same manner as any other attribute value when the name is known. When you perform a get with the XPath “@<name>” , the attribute value will be returned. If the name is unknown , the values can be iterated and accessed one by one. The following example code shows a case where the attribute name is known:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyAttrOnlyMixed"
 targetNamespace="http://AnyAttrOnly">
  <xsd:complexType name="AnyAttrOnly">
    <xsd:sequence>
      <xsd:element name="element" type="xsd:string"/>
    </xsd:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://GlobalAttrs">
  <xsd:attribute name="globalAttribute" type="xsd:string"/>
</xsd:schema>

 DataObject dobj = ...

 // Get the global attribute Property that is going to be set
 Property globalProp = boXsdHelper.getGlobalProperty("http://GlobalAttrs",
 "globalAttribute", false);

 // Set the value on the dobj, just like any other data
 dobj.set(globalProp, "foo");

 // The data can now be accessed by a get call
 System.out.println(dobj.get("@globalAttribute")); // Displays "foo"
```

Valid mappings for data in an anyAttribute:

The AnyAttribute tag is similar to the any tag, and is a set of name/value pairs. Consequently, the only valid mapping for anyAttribute is another anyAttribute.

Individually, the values contained in the anyAttribute data are basic simple types that follow all the rules of simple type mapping.

Configuring logging properties for business objects

Use the administrative console to enable logging and to select the required log specification, level of detail, and output format of the log. Business objects provide different log specifications for different functionality areas, and selecting the appropriate level of logging helps isolate and troubleshoot the problems in that area.

About this task

You can change the log configuration statically or dynamically. Static configuration takes effect when you start or restart the application server. Dynamic, or runtime, configuration changes are applied immediately.

To enable logging and set the output properties for a log, use the following procedure.

Procedure

1. In the navigation pane of the administrative console, click **Servers** > **Server Types** > **WebSphere application servers**.
2. Click the name of the server that you want to work with.
3. Under **Troubleshooting**, click **Logs and trace**.
4. Click **Change Log Detail Levels**.
5. Specify when you want the change to take effect:
   - For a static change to the configuration, click the **Configuration** tab.
   - For a dynamic change to the configuration, click the **Runtime** tab.
6. Expand the list, and then click the names of the public packages (described in Table 66) whose logging level you want to modify.

Table 66. Logging specification descriptions

Logging specification	Usage
BOChangeSummary	Enable this log specification in the following circumstances: • When changes are not logged in a business graph even though you have change summary enabled • You are using the BOChangeSummary API
BOCopy	Enable this log specification if you are using the BOCopy API in a business application and you notice that the copied object does not match the original object.
BOCore	Enable this log specification for any issues related to initialization of Business Object framework, loading of Business Object services, or schema loading.
BODataObject	Enable this log specification if you use the BODataObject API to access data object aspects of a business object (such as the change summary, event summary, and business graph). Also enable this log specification when you are using the BOLAZY parsing mode and you find issues related to retrieving, modifying, or unsetting the data on the business object.
BOEquality	Enable this log specification if you find issues when using the BOEquality API to determine if two business objects are equal.
BOFactory	Enable this log specification if you find issues when creating business objects using the BOFactory API.
BOImpl	Enable this log specification for issues related to other Business Object framework implementation classes.
BOInstanceValidator	Enable this log specification for issues related to business object instance validation.

Table 66. Logging specification descriptions (continued)

Logging specification	Usage
BOType	Enable this log specification for issues that arise when using the BOType API to create types.
BOTypeMetadata	Enable this log specification for issues related to transforming an annotation string to a data object (or vice-versa) using the BOTypeMetadata API.
BOXMLSerializer or BOXMLSerializerXMLTrace	Enable these log specifications for any issues related to serialization or deserialization of business objects. Enabling BOXMLSerializerXMLTrace will print the XML document being read or written in the log file.

For any other issues, enable the BOCore log specification.

Note: You can enable multiple log details for issues spanning multiple areas.

7. Expand **Message and Trace Levels**, and then select the logging level (as described in Table 67).

Table 67. Logging levels

Logging Level	Description
Fatal	The task cannot continue or the component cannot function.
Severe	The task cannot continue, but the component can still function. This logging level also includes conditions that indicate an impending fatal error (that is, situations that strongly suggest that resources are on the verge of being depleted).
Warning	A potential error has occurred or a severe error is impending. This logging level also includes conditions that indicate a progressive failure (for example, the potential leaking of resources).
Audit	A significant event has occurred that affects the server state or resources.
Info	The task is running. This logging level includes general information outlining the overall progress of a task.
Config	The status of a configuration is reported or a configuration change has occurred.
Detail	The subtask is running. This logging level includes general information detailing the progress of a subtask.
All	Superset of all the logging levels.

8. Click **Apply**.
9. Click **OK**.

Results

- If you selected dynamic configuration, the log files immediately contain information about business objects that are processed by the server.
- If you chose to static configuration, the log files will begin logging the same information after you restart your server.

What to do next

To have static configuration changes take effect, stop and then restart the process server.

BusinessObject.properties file

The BusinessObject.properties file is used to configure the runtime behavior of the Business Object Framework.

The file is present in *install_root*/plugins/com.ibm.ws.bpm.bo.emf.jar file. The following list of parameters is defined in this file and can be modified to adjust the runtime behavior. The should modify the default value with caution and only if it is necessary for a given environment.

Parameters

WAS_XSDATE_BEHAVIOR
This property defines the way that timezones are handled for date type properties. During deserialization and serialization, a timezone must be assumed for date typed properties. A value of TRUE sets the assumed timezone to be the default local timezone while a value of FALSE sets the assumed timezone to be GMT timezone and normalize the values to GMT.

CS_LOGGING_SERIALIZE
This property defines whether a logging state ChangeSummary is serialized along with the BusinessGraph so that it is persisted across a serialize/deserialize cycle. A value of TRUE means that the logging state is persisted across serialization. A value of FALSE means that the logging state is reset to the default logging state on a deserialization.

RELAXED_TYPE_CHECKING
The default value for this parameter is FALSE. There are some scenarios that result in duplicate type definitions being loaded into packages. These conditions are errors that should be reported and resolved, but in some cases it is possible to continue by relaxing the type checking to be less strict and allow the duplicate types to compare as equal. For these cases, you can set this value to TRUE, but you should analyze and fix in problems before deploying your applications in a production environment.

VERSION
This property defines the behavior of business objects in the system, allowing you to restrict business object framework behavioral changes that would otherwise cause functional differences in an application. The business object framework behavior changes can be disabled until the applications are updated to adapt the behavioral changes. The following table lists the specific behavioral changes:

Table 68. Scenarios where the VERSION parameter will cause behavior changes

Scenario	Value = "*1*"	Value = "*2*"
DataObject.isSet() API	Throws runtime exceptions	Return true or false
DataObject.get<T>(String) API with invalid XPaths	Throws runtime exceptions	Return null
ChangeSummary.getOldValues() API when there are no changes	Returns null	Returns an empty list
DataObject.get/set<T>(int)	Uses underlying model index	Uses property index

Table 68. Scenarios where the VERSION parameter will cause behavior changes (continued)

Scenario	Value = "1"	Value = "2"
`setDate()` on property of type `dDay`, `gMonth`, `gMonthDay`, or `gTime`,	Value set is `java,util.Date.toString()`	Value set is corresponding part of the Date value
`BOEquality` API	Does not properly compare `gYear`, `gYearMonth`, `date`, or `dateTime`	Compares all date values as specified in the XSD.

TRUNCATE_NAMESPACE_PREFIX
This property defines if the namespace prefix is truncated or not. The default value of TRUE truncates the namespace prefix during serialization. A value of FALSE would not perform the truncation.

SERIALIZATION_SCHEMA_COMPLIANCE
The property defines the serialization behavior of business object framework when working with business objects with any type property. If the value is set to FALSE (default), the serialized XML would contain the properties in the order in which they were set in the business object. The order does not comply with the sequence definition in the schema. If the value is set to TRUE the serialized XML would comply with the sequence definition in the schema. The order in which the properties were set does not matter.

Widget programming

WebSphere ESB provides a set of widgets for use in Business Space. These widgets are enabled for a process management context. You can extend these widgets, and also create your own widgets, possibly integrating them with widgets from IBM.

Bindings

At the core of a service-oriented architecture is the concept of a *service*, a unit of functionality accomplished by an interaction between computing devices. An *export* defines the external interface (or access point) of a module, so that Service Component Architecture (SCA) components within the module can provide their services to external clients. An *import* defines an interface to services outside a module, so the services can be called from within the module. You use protocol-specific *bindings* with imports and exports to specify the means of transporting the data into or out of the module.

Exports

External clients can invoke SCA components in an integration module over a variety of protocols (such as HTTP, JMS, MQ, and RMI/IIOP) with data in a variety of formats (such as XML, CSV, COBOL, and JavaBeans). Exports are components that receive these requests from external sources and then invoke WebSphere ESB components using the SCA programming model.

For example, in the following figure, an export receives a request over the HTTP protocol from a client application. The data is transformed into a business object, the format used by the SCA component. The component is then invoked with that data object.

Figure 267. An export with HTTP binding

Imports

An SCA component might want to invoke a non-SCA external service that expects data in a different format. An import is used by the SCA component to invoke the external service using the SCA programming model. The import then invokes the target service in the way that the service expects.

For example, in the following figure, a request from an SCA component is sent, by the import, to an external service. The business object, which is the format used by the SCA component, is transformed to the format expect by the service, and the service is invoked.

Figure 268. An import with HTTP binding

List of bindings

You use IBM Integration Designer to generate a binding for an import or export and to configure the binding. The types of bindings that are available are described in the following list.

- SCA

  The SCA binding, which is the default, lets your service communicate with services in other SCA modules. You use an import with an SCA binding to access a service in another SCA module. You use an export with an SCA binding to offer a service to other SCA modules.

- Web service

  A Web service binding lets you access an external service using interoperable SOAP messages and qualities of service. You can also use Web service bindings to include attachments as part of the SOAP message.

  The Web service binding can use a transport protocol of either SOAP/HTTP (SOAP over HTTP) or SOAP/JMS (SOAP over JMS). Regardless of the transport (HTTP or JMS) used to convey the SOAP messages, Web service bindings always handle request/response interactions synchronously.

- HTTP

  The HTTP binding lets you access an external service using the HTTP protocol, where non-SOAP messages are used, or where direct HTTP access is required. This binding is used when you are working with Web services that are based on the HTTP model (that is, services that use well-known HTTP interface operations such as GET, PUT, DELETE, and so on).
- Enterprise JavaBeans (EJB)

  The EJB bindings let SCA components interact with services provided by Java EE business logic running on a Java EE server.
- EIS

  The EIS (enterprise information system) binding, when used with a JCA resource adapter, lets you access services on an enterprise information system or make your services available to the EIS.
- JMS bindings

  Java Message Service (JMS), generic JMS, and WebSphere MQ JMS (MQ JMS) bindings are used for interactions with messaging systems, where asynchronous communication through message queues is critical for reliability.

  An export with one of the JMS bindings watches a queue for the arrival of a message and asynchronously sends the response, if any, to the reply queue. An import with one of the JMS bindings builds and sends a message to a JMS queue and watches a queue for the arrival of the response, if any.
  - JMS

    The JMS binding lets you access the WebSphere-embedded JMS provider.
  - Generic JMS

    The generic JMS binding lets you access a non-IBM vendor messaging system.
  - MQ JMS

    The MQ JMS binding lets you access the JMS subset of a WebSphere MQ messaging system. You would use this binding when the JMS subset of functions is sufficient for your application.
- MQ

  The WebSphere MQ binding lets you communicate with MQ native applications, bringing them into the service oriented architecture framework and providing access to MQ-specific header information. You would use this binding when you need to use MQ native functions.

Export and import binding overview

An export lets you make services in an integration module available to external clients, and an import makes it possible for your SCA components in an integration module to call external services. The binding associated with the export or import specifies the relationship between protocol messages and business objects. It also specifies the way that operations and faults are selected.

Flow of information through an export

An export receives a request, which is intended for the component to which the export is wired, over a specific transport determined by the associated binding (for example, HTTP).

Figure 269. Flow of a request through the export to a component

When the export receives the request, the following sequence of events occurs:

1. For WebSphere MQ bindings only, the header data binding transforms the protocol header into a header data object.
2. The function selector determines the native method name from the protocol message. The native method name is mapped by the export configuration to the name of an operation on the interface of the export.
3. The request data handler or data binding on the method transforms the request to a request business object.
4. The export invokes the component method with the request business object.
   - The HTTP export binding, the Web service export binding, and the EJB export binding invoke the SCA component synchronously.
   - The JMS, Generic JMS, MQ JMS, and WebSphere MQ export bindings invoke the SCA component asynchronously.

Note that an export can propagate the headers and user properties it receives over the protocol, if context propagation is enabled. Components that are wired to the export can then access these headers and user properties. See the “Propagation” topic in the WebSphere Integration Developer information center for more information.

If this is a two-way operation, the component returns a response.

Figure 270. Flow of a response back through the export

The following sequence of steps occurs:

1. If a normal response message is received by the export binding, the response data handler or data binding on the method transforms the business object to a response.

   If the response is a fault, the fault data handler or data binding on the method transforms the fault to a fault response.

   For HTTP export bindings only, if the response is a runtime exception, the runtime exception data handler, if configured, is called.
2. For WebSphere MQ bindings only, the header data binding transforms the header data objects into protocol headers.
3. The export sends the service response over the transport.

Flow of information through an import

Components send requests to services outside the module using an import. The request is sent, over a specific transport determined by the associated binding.

Figure 271. Flow from a component through the import to a service

The component invokes the import with a request business object.

Note:

- The HTTP import binding, the Web service import binding, and the EJB import binding should be invoked synchronously by the calling component.
- The JMS, Generic JMS, MQ JMS, and WebSphere MQ import binding should be invoked asynchronously.

After the component invokes the import, the following sequence of events occurs:

1. The request data handler or data binding on the method transforms the request business object into a protocol request message.
2. For WebSphere MQ bindings only, the header data binding on the method sets the header business object in the protocol header.
3. The import invokes the service with the service request over the transport.

If this is a two-way operation, the service returns a response, and the following sequence of steps occurs:

Figure 272. Flow of a response back through the import

1. For WebSphere MQ bindings only, the header data binding transforms the protocol header into a header data object.
2. A determination is made about whether the response is a fault.
   - If the response is a fault, the fault selector inspects the fault to determine which WSDL fault it maps to. The fault data handler on the method then transforms the fault to a fault response.
   - If the response is a runtime exception, the runtime exception data handler, if configured, is called.
3. The response data handler or binding on the method transforms the response to a response business object.
4. The import returns the response business object to the component.

Export and import binding configuration

One of the key aspects of export and import bindings is data format transformation, which indicates how data is mapped (deserialized) from a native wire format to a business object or how it is mapped (serialized) from a business object to a native wire format. For bindings associated with exports, you can also specify a function selector to indicate which operation should be performed on the data. For bindings associated with exports or imports, you can indicate how faults that occur during processing should be handled.

In addition, you specify transport-specific information on bindings. For example, for an HTTP binding, you specify the endpoint URL. For the HTTP binding, the transport-specific information is described in the “Generating an HTTP import binding” and “Generating an HTTP export binding” topics. You can also find information about other bindings in the information center.

Data format transformation in imports and exports

When an export or import binding is configured in WebSphere Integration Developer, one of the configuration properties that you specify is the data format used by the binding.

- For export bindings, in which a client application sends requests to and receives responses from an SCA component, you indicate the format of the native data. Depending on the format, the system selects the appropriate data handler or data binding to transform the native data to a business object (which is used by the SCA component) and conversely to transform the business object to native data (which is the response to the client application).
- For import bindings, in which an SCA component sends requests to and receives responses from a service outside the module, you indicate the data format of the native data. Depending on the format, the system selects the appropriate data handler or data binding to transform the business object to native data and vice versa.

WebSphere ESB provides a set of predefined data formats and corresponding data handlers or data bindings that support the formats. You can also create your own custom data handlers and register the data format for those data handlers. For more information, see the "Developing data handlers" topic in the WebSphere Integration Developer information center.

- *Data handlers* are protocol-neutral and transform data from one format to another. In WebSphere ESB, data handlers typically transform native data (such as XML, CSV, and COBOL) to a business object and a business object to native data. Because they are protocol-neutral, you can reuse the same data handler with a variety of export and import bindings. For example, you can use the same XML data handler with an HTTP export or import binding or with a JMS export or import binding.
- *Data bindings* also transform native data to a business object (and vice versa), but they are protocol-specific. For example, an HTTP data binding can be used with an HTTP export or import binding only. Unlike data handlers, an HTTP data binding cannot be reused with an MQ export or import binding.

  Note: Three HTTP data bindings (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML, and HTTPServiceGatewayDataBinding) are deprecated as of WebSphere ESB Version 7.0. Use data handlers whenever possible.

As noted earlier, you can create custom data handlers, if necessary. You can also create custom data bindings; however, it is recommended that you create custom data handlers because they can be used across multiple bindings.

Data handlers:

Data handlers are configured against export and import bindings to transform data from one format to another in a protocol-neutral fashion. Several data handlers are provided as part of the product, but you can also create your own data handler, if necessary. You can associate a data handler with an export or import binding at one of two levels: you can associate it with all operations in the interface of the export or import, or you can associate it with a specific operation for the request or response.

Predefined data handlers

You use IBM Integration Designer to specify the data handler that you want to use.

The data handlers that are predefined for your use are listed in the following table, which also describes how each data handler transforms inbound and outbound data.

Note: Except where noted, these data handlers can be used with JMS, Generic JMS, MQ JMS, WebSphere MQ, and HTTP bindings.
See the "Data handlers" topic in the Integration Designer information center for more detailed information.

Table 69. Predefined data handlers

Data handler	Native data to business object	Business object to native data
ATOM	Parses ATOM feeds into an ATOM feed business object.	Serializes an ATOM feed business object to ATOM feeds.
Delimited	Parses delimited data into a business object.	Serializes a business object to delimited data, including CSV.
Fixed Width	Parses fixed-width data into a business object.	Serializes a business object to fixed-width data.
Handled by WTX	Delegates data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is derived by the data handler.	Delegates data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is derived by the data handler.
Handled by WTX Invoker	Delegates the data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is supplied by the user.	Delegates the data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is supplied by the user.
JAXB	Serializes Java beans to a business object using the mapping rules defined by the Java Architecture for XML Binding (JAXB) specification.	Deserializes a business object to Java beans using the mapping rules defined by the JAXB specification.
JAXWS **Note:** The JAXWS data handler can be used only with the EJB binding.	Used by an EJB binding to transform a response Java object or exception Java object to a response business object using the mapping rules defined by the Java API for XML Web Services (JAX-WS) specification.	Used by an EJB binding to transform a business object to the outgoing Java method parameters using the mapping rules defined by the JAX-WS specification.
JSON	Parses JSON data into a business object.	Serializes a business object to JSON data.
Native body	Parses the native bytes, text, map, stream, or object into one of five base business objects (text, bytes, map, stream, or object).	Transforms the five base business objects into byte, text, map, stream, or object.
SOAP	Parses the SOAP message (and the header) into a business object.	Serializes a business object to a SOAP message.
XML	Parses XML data into a business object.	Serializes a business object to XML data.
UTF8XMLDataHandler	Parses UTF-8 encoded XML data into a business object.	Serializes a business object into UTF-8 encoded XML data when sending a message.

Creating a data handler

Detailed information about creating a data handler can be found in the "Developing data handlers" topic in the Integration Designer information center.

Data bindings:

Data bindings are configured against export and import bindings to transform data from one format to another. Data bindings are specific to a protocol. Several data bindings are provided as part of the product, but you can also create your own data binding, if necessary. You can associate a data binding with an export or import binding at one of two levels–you can associate it with all operations in the interface of the export or import, or you can associate it with a specific operation for the request or response.

You use WebSphere Integration Developer to specify which data binding you want to use or to create your own data binding. A discussion of creating data bindings can be found in the "Overview of JMS, MQ JMS and generic JMS bindings" section of the WebSphere Integration Developer information center.

JMS bindings

The following table lists the data bindings that can be used with:

- JMS bindings
- Generic JMS bindings
- WebSphere MQ JMS bindings

The table also includes a description of the tasks that the data bindings perform.

Table 70. Predefined data bindings for JMS bindings

Data binding	Native data to business object	Business object to native data
Serialized Java object	Transforms the Java serialized object into a business object (which is mapped as the input or output type in the WSDL).	Serializes a business object to the Java serialized object in the JMS object message.
Wrapped bytes	Extracts the bytes from the incoming JMS bytes message and wraps them into the JMSBytesBody business object.	Extracts the bytes from the JMSBytesBody business object and wraps them into the outgoing JMS bytes message
Wrapped map entry	Extracts the name, value, and type information for every entry in the incoming JMS map message and creates a list of MapEntry business objects. It then wraps the list into the JMSMapBody business object	Extracts the name, value, and type information from the MapEntry list in the JMSMapBody business object and creates the corresponding entries in the outgoing JMS map message.
Wrapped object	Extracts the object from the incoming JMS object message and wraps it into the JMSObjectBody business object.	Extracts the object from the JMSObjectBody business object and wraps it into the outgoing JMS object message.
Wrapped text	Extracts the text from the incoming JMS text message and wraps it into the JMSTextBody business object.	Extracts the text from the JMSTextBody business object and wraps it into the outgoing JMS text message.

WebSphere MQ bindings

The following table lists the data bindings that can be used with WebSphere MQ and describes the tasks that the data bindings perform.

Table 71. Predefined data bindings for WebSphere MQ bindings

Data binding	Native data to business object	Business object to native data
Serialized Java object	Transforms the Java serialized object from the incoming message into a business object (which is mapped as the input or output type in the WSDL).	Transforms a business object to the Java serialized object in the outgoing message
Wrapped bytes	Extracts the bytes from the unstructured MQ bytes message and wraps them into the JMSBytesBody business object.	Extracts the bytes from a JMSBytesBody business object and wraps the bytes into the outgoing unstructured MQ bytes message.
Wrapped text	Extracts the text from an unstructured MQ text message and wraps it into a JMSTextBody business object.	Extracts text from a JMSTextBody business object and wraps it in an unstructured MQ text message.
Wrapped stream entry	Extracts the name and type information for every entry in the incoming JMS stream message and creates a list of the StreamEntry business objects. It then wraps the list into the JMSStreamBody business object.	Extracts the name and type information from the StreamEntry list in the JMSStreamBody business object and creates corresponding entries in the outgoing JMSStreamMessage.

In addition to the data bindings listed in Table 5 on page 62, WebSphere MQ also uses header data bindings. See the WebSphere Integration Developer information center for details.

HTTP bindings

The following table lists the data bindings that can be used with HTTP and describes the tasks that the data bindings perform.

Table 72. Predefined data bindings for HTTP bindings

Data binding	Native data to business object	Business object to native data
Wrapped bytes	Extracts the bytes from the body of the incoming HTTP message and wraps them into the HTTPBytes business object.	Extracts the bytes from the HTTPBytes business object and adds them to the body of the outgoing HTTP message.
Wrapped text	Extracts the text from the body of the incoming HTTP message and wraps it into the HTTPText business object.	Extracts the text from the HTTPText business object and adds it to the body of the outgoing HTTP message.

Function selectors in export bindings

A function selector is used to indicate which operation should be performed on the data for a request message. Function selectors are configured as part of an export binding.

Consider an SCA export that exposes an interface. The interface contains two operations–Create and Update. The export has a JMS binding that reads from a queue.

When a message arrives on the queue, the export is passed the associated data, but which operation from the export's interface should be invoked on the wired component? The operation is determined by the function selector and the export binding configuration.

The function selector returns the native function name (the function name in the client system that sent the message). The native function name is then mapped to the operation or function name on the interface associated with the export. For example, in the following figure, the function selector returns the native function name (CRT) from the incoming message, the native function name is mapped to the Create operation, and the business object is sent to the SCA component with the Create operation.

Figure 273. The function selector

If the interface has only one operation, there is no need to specify a function selector.

Several prepackaged function selectors are available and are listed in the sections that follow.

JMS bindings

The following table lists the function selectors that can be used with:

- JMS bindings
- Generic JMS bindings
- WebSphere MQ JMS bindings

Table 73. Predefined function selectors for JMS bindings

Function selector	Description
JMS function selector for simple JMS data bindings	Uses the JMSType property of the message to select the operation name.
JMS header property function selector	Returns the value of the JMS String Property, TargetFunctionName, from the header.

Table 73. Predefined function selectors for JMS bindings (continued)

Function selector	Description
JMS service gateway function selector	Determines if the request is a one-way or two-way operation by examining the JMSReplyTo property set by the client.

WebSphere MQ bindings

The following table lists the function selectors that can be used with WebSphere MQ bindings.

Table 74. Predefined function selectors for WebSphere MQ bindings

Function selector	Description
MQ handleMessage function selector	Returns handleMessage as a value, which is mapped using the export method bindings to the name of an operation on the interface.
MQ uses JMS default function selector	Reads the native operation from the TargetFunctionName property of the folder of an MQRFH2 header.
MQ uses message body format as native function	Finds the Format field of the last header and returns that field as a String.
MQ type function selector	Creates a method in your export binding by retrieving a URL containing the Msd, Set, Type and Format properties found in the MQRFH2 header.
MQ service gateway function selector	Uses the MsgType property in the MQMD header to determine the operation name.

HTTP bindings

The following table lists the function selectors that can be used with HTTP bindings.

Table 75. Predefined function selectors for HTTP bindings

Function selector	Description
HTTP function selector based on the TargetFunctionName header	Uses the TargetFunctionName HTTP header property from the client to determine which operation to invoke at runtime from the export.
HTTP function selector based on the URL and HTTP method	Uses the relative path from the URL appended with the HTTP method from the client to determine the native operation defined on the export.
HTTP service gateway function selector based on URL with an operation name	Determines the method to invoke based on the URL if "operationMode = oneWay" has been appended to the request URL.

Note: You can also create your own function selector, using IBM Integration Designer. Information about creating a function selector is provided in the IBM Integration Designer information center. For example, a description of creating a function selector for WebSphere MQ bindings can be found in “Overview of the MQ function selectors”.

Fault handling

You can configure your import and export bindings to handle faults (for example, business exceptions) that occur during processing by specifying fault data handlers. You can set up a fault data handler at three levels–you can associate a fault data handler with a fault, with an operation, or for all operations with a binding.

A fault data handler processes fault data and transforms it into the correct format to be sent by the export or import binding.

- For an export binding, the fault data handler transforms the exception business object sent from the component to a response message that can be used by the client application.
- For an import binding, the fault data handler transforms the fault data or response message sent from a service into an exception business object that can be used by the SCA component.

For import bindings, the binding calls the fault selector, which determines whether the response message is a normal response, a business fault, or a runtime exception.

You can specify a fault data handler for a particular fault, for an operation, and for all operations with a binding.

- If the fault data handler is set at all three levels, the data handler associated with a particular fault is called.
- If fault data handlers are set at the operation and binding levels, the data handler associated with the operation is called.

Two editors are used in IBM Integration Designer to specify fault handling. The interface editor is used to indicate whether there will be a fault on an operation. After a binding is generated with this interface, the editor in the properties view lets you configure how the fault will be handled. For more information, see the "Fault selectors" topic in the IBM Integration Designer information center.

How faults are handled in export bindings:

When a fault occurs during the processing of the request from a client application, the export binding can return the fault information to the client. You configure the export binding to specify how the fault should be processed and returned to the client.

You configure the export binding using IBM Integration Designer.

During request processing, a client invokes an export with a request, and the export invokes the SCA component. During the processing of the request, the SCA component can either return a business response or can throw a service business exception or a service runtime exception. When this occurs, the export binding transforms the exception into a fault message and sends it to the client, as shown in the following figure and described in the sections that follow.

Figure 274. How fault information is sent from the component through the export binding to the client

You can create a custom data handler or data binding to handle faults.

Business faults

Business faults are business errors or exceptions that occur during processing.

Consider the following interface, which has a createCustomer operation on it. This operation has two business faults defined: CustomerAlreadyExists and MissingCustomerId.

Figure 275. Interface with two faults

In this example, if a client sends a request to create a customer (to this SCA component) and that customer already exists, the component throws a CustomerAlreadyExists fault to the export. The export needs to propagate this business fault back to the calling client. To do so, it uses the fault data handler that is set up on the export binding.

When a business fault is received by the export binding, the following processing occurs:

1. The binding determines which fault data handler to invoke for handling the fault. If the service business exception contains the fault name, the data handler that is set up on the fault is called. If the service business exception does not contain the name of the fault, the fault name is derived by matching the fault types.
2. The binding calls the fault data handler with the data object from the service business exception.
3. The fault data handler transforms the fault data object to a response message and returns it to the export binding.
4. The export returns the response message to the client.

If the service business exception contains the fault name, the data handler that is set up on the fault is called. If the service business exception does not contain the

name of the fault, the fault name is derived by matching the fault types.

Runtime exceptions

A runtime exception is an exception that occurs in the SCA application during the processing of a request that does not correspond to a business fault. Unlike business faults, runtime exceptions are not defined on the interface.

In certain scenarios, you might want to propagate these runtime exceptions to the client application so that the client application can take the appropriate action.

For example, if a client sends a request (to the SCA component) to create a customer and an authorization error occurs during processing of the request, the component throws a runtime exception. This runtime exception has to be propagated back to the calling client so it can take the appropriate action regarding the authorization. This is achieved by the runtime exception data handler configured on the export binding.

Note: You can configure a runtime exception data handler only on HTTP bindings.

The processing of a runtime exception is similar to the processing of a business fault. If a runtime exception data handler was set up, the following processing occurs:

1. The export binding calls the appropriate data handler with the service runtime exception.
2. The data handler transforms the fault data object to a response message and returns it to the export binding.
3. The export returns the response message to the client.

Fault handling and runtime exception handling are optional. If you do not want to propagate faults or runtime exceptions to the calling client, do not configure the fault data handler or runtime exception data handler.

How faults are handled in import bindings:

A component uses an import to send a request to a service outside the module. When a fault occurs during the processing of the request, the service returns the fault to the import binding. You can configure the import binding to specify how the fault should be processed and returned to the component.

You configure the import binding using IBM Integration Designer. You can specify a fault data handler (or data binding), and you also specify a fault selector.

Fault data handlers

The service that processes the request sends, to the import binding, fault information in the form of an exception or a response message that contains the fault data.

The import binding transforms the service exception or response message into a service business exception or service runtime exception, as shown in the following figure and described in the sections that follow.

Figure 276. How fault information is sent from the service through the import to the component

You can create a custom data handler or data binding to handle faults.

Fault selectors

When you configure an import binding, you can specify a fault selector. The fault selector determines whether the import response is an actual response, a business exception, or a runtime fault. It also determines, from the response body or header, the native fault name, which is mapped by the binding configuration to the name of a fault in the associated interface.

Two types of prepackaged fault selectors are available for use with JMS, MQ JMS, Generic JMS, WebSphere MQ, and HTTP imports:

Table 76. Prepackaged fault selectors

Fault selector type	Description
Header-based	Determines whether a response message is a business fault, a runtime exception, or a normal message based on the headers in the incoming response message.
SOAP	Determines whether the response SOAP message is a normal response, business fault, or runtime exception.

The following shows examples of header-based fault selectors and the SOAP fault selector.

- Header-based fault selector

  If an application wants to indicate that the incoming message is a business fault, there must be two headers in the incoming message for business faults, which is shown as follows:

  ```
  Header name = FaultType, Header value = Business
  Header name = FaultName, Header value = <user defined native fault name>
  ```

  If an application wants to indicate that the incoming response message is a runtime exception, then there must be one header in the incoming message, which is shown as follows:

  ```
  Header name = FaultType, Header value = Runtime
  ```

- SOAP fault selector

  A business fault can be sent as part of the SOAP message with the following custom SOAP header. "CustomerAlreadyExists" is the name of the fault in this case.

  ```
  <ibmSoap:BusinessFaultName
  xmlns:ibmSoap="http://www.ibm.com/soap">CustomerAlreadyExists
  <ibmSoap:BusinessFaultName>
  ```

The fault selector is optional. If you do not specify a fault selector, the import binding cannot determine the type of response. The binding therefore treats it as a business response and calls the response data handler or data binding.

You can create a custom fault selector. The steps for creating a custom fault selector are provided in the "Developing a custom fault selector" topic of the IBM Integration Designer information center.

Business faults

A business fault can occur when there is an error in the processing of a request. For example, if you send a request to create a customer and that customer already exists, the service sends a business exception to the import binding.

When a business exception is received by the binding, the processing steps depend on whether a fault selector has been set up for the binding.

- If no fault selector was set up, the binding calls the response data handler or data binding.
- If a fault selector was set up, the following processing occurs:
  1. The import binding calls the fault selector to determine whether the response is business fault, business response, or runtime fault.
  2. If the response is a business fault, the import binding calls the fault selector to provide the native fault name.
  3. The import binding determines the WSDL fault corresponding to the native fault name returned by the fault selector.
  4. The import binding determines the fault data handler that is configured for this WSDL fault.
  5. The import binding calls this fault data handler with the fault data.
  6. The fault data handler transforms the fault data to a data object and returns it to the import binding.
  7. The import binding constructs a service business exception object with the data object and the fault name.
  8. The import returns the service business exception object to the component.

Runtime exceptions

A runtime exception can occur when there is a problem in communicating with the service. The processing of a runtime exception is similar to the processing of a business exception. If a fault selector was set up, the following processing occurs:

1. The import binding calls the appropriate runtime exception data handler with the exception data.
2. The runtime exception data handler transforms the exception data to a service runtime exception object and returns it to the import binding.
3. The import returns the service runtime exception object to the component.

Interoperability between SCA modules and Open SCA services

The IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture (SCA) provides a simple, yet powerful programming model for constructing applications based on the Open SCA specifications. The SCA modules of WebSphere ESB use import and export bindings to interoperate with Open SCA

services developed in a Rational Application Developer environment and hosted by the WebSphere Application Server Feature Pack for Service Component Architecture.

An SCA application invokes an Open SCA application by way of an import binding. An SCA application receives a call from an Open SCA application by way of an export binding. A list of supported bindings is shown in "Invoking services over interoperable bindings" on page 71.

Invoking Open SCA services from SCA modules

SCA applications developed with IBM Integration Designer can invoke Open SCA applications developed in a Rational Application Developer environment. This section provides an example of invoking an Open SCA service from an SCA module using an SCA import binding.

Figure 277. Component in SCA module invoking Open SCA service

No special configuration is required to invoke an Open SCA service.

To connect to an Open SCA service by way of an SCA import binding, you provide the component name and service name of the Open SCA service in the import binding.

1. To obtain the name of the target component and service from the Open SCA composite, perform the following steps:
   a. Ensure that the **Properties** tab is open by clicking **Window** > **Show View** > **Properties**.
   b. Open the composite editor by double-clicking the composite diagram that contains the component and service. For example, for a component named **customer**, the composite diagram is **customer.composite_diagram**.
   c. Click the target component.
   d. In the **Name** field of the **Properties** tab, note the name of the target component.
   e. Click the service icon associated with the component.
   f. In the **Name** field of the **Properties** tab, note the name of the service.
2. To configure the WebSphere ESB import to connect it to the Open SCA service, perform the following steps:
   a. In IBM Integration Designer, navigate to the **Properties** tab of the SCA import that you want to connect to the Open SCA service.
   b. In the **Module name** field, enter the component name from step 1d on page 70.
   c. In the **Export name** field, enter the service name from step 1f on page 70.
   d. Save your work by pressing Ctrl+S.

Invoking SCA modules from Open SCA services

Open SCA applications developed in a Rational Application Developer environment can invoke SCA applications developed with IBM Integration Designer. This section provides an example of invoking an SCA module (by way of an SCA export binding) from an Open SCA service.

Figure 278. Open SCA service invoking component in SCA module

To connect to an SCA component by way of an Open SCA reference binding, you provide the module name and export name.

1. To obtain the name of the target module and export, perform the following steps:
   a. In IBM Integration Designer, open the module in the assembly editor by double-clicking the module.
   b. Click the export.
   c. In the **Name** field of the **Properties** tab, note the name of the export.
2. Configure the Open SCA reference that you want to connect to the WebSphere ESB module and export:
   a. In Rational Application Developer, open the composite editor by double-clicking the composite diagram that contains the component and service.
   b. Click the reference icon of the component reference to display the reference properties in the **Properties** tab.
   c. Click the **Binding** tab on the left side of the page.
   d. Click **Bindings** and then click **Add**.
   e. Select the **SCA** binding.
   f. In the **Uri** field, enter the WebSphere ESB module name, followed by a slash ("/"), followed by the export name (which you determined in step 1c on page 71).
   g. Click **OK**.
   h. Save your work by pressing Ctrl+S.

Invoking services over interoperable bindings

The following bindings are supported for interoperability with an Open SCA service.

- SCA binding

  In WebSphere ESB, when an SCA module invokes an Open SCA service by way of an SCA import binding, the following invocation styles are supported:
  - Asynchronous (one-way)
  - Synchronous (request/response)

The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

Note that the SCA binding supports transaction and security context propagation.

- Web service (JAX-WS) binding with either the SOAP1.1/HTTP or SOAP1.2/HTTP protocol

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  In addition, the following qualities of service are supported:

  – Web Services Atomic Transaction
  – Web Services Security

- EJB binding

  A Java interface is used to define the interaction between an SCA module and an Open SCA service when the EJB binding is used.

  Note that the EJB binding supports transaction and security context propagation.

- JMS bindings

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  The following JMS providers are supported:

  – WebSphere Platform Messaging (JMS Binding)
  – WebSphere MQ (MQ JMS Binding)

Note: Business Graphs are not interoperable across any SCA bindings and, therefore, are not supported in interfaces used to interoperate with the WebSphere Application Server Feature Pack for Service Component Architecture.

Binding types

You use protocol-specific *bindings* with imports and exports to specify the means of transporting data into or out of a module.

Selecting appropriate bindings

When you are creating an application, you need to know how to select the binding that is most appropriate to the needs of your application.

The bindings available in IBM Integration Designer provide a range of choices. In this list, you can determine which type of binding might be most suitable for the needs of your application.

Consider a *Service Component Architecture (SCA)* binding when these factors are applicable:

- All services are contained in modules; that is, there are no external services.
- You want to separate function into different SCA modules that interact directly with each other.
- The modules are tightly coupled.

Consider a *Web Service* binding when these factors are applicable:

- You need to access an external service over the Internet or provide a service over the Internet.
- The services are loosely coupled.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

- The protocol of the external service you are accessing or the service you want to provide is SOAP/HTTP or SOAP/JMS.

Consider an *HTTP* binding when these factors are applicable:

- You need to access an external service over the Internet or provide a service over the Internet and you are working with other web services such as GET, PUT, and DELETE.
- The services are loosely coupled.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

Consider an *Enterprise JavaBeans (EJB)* binding when these factors are applicable:

- The binding is for an imported service that is itself an EJB or that needs to be accessed by EJB clients.
- The imported service is loosely coupled.
- Stateful EJB interactions are not required.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

Consider an *Enterprise Information Systems (EIS)* binding when these factors are applicable:

- You need to access a service on an EIS system using a resource adapter.
- Synchronous data transmission is preferred over asynchronous.

Consider a *Java Message Service (JMS)* binding when these factors are applicable:

Important: There are several types of JMS bindings. If you expect to exchange SOAP messages using JMS, consider the Web service binding with the SOAP/JMS protocol. See "Web service bindings" on page 74.

- You need to access a messaging system.
- The services are loosely coupled.
- Asynchronous data transmission is preferred over synchronous.

Consider a *Generic Java Message Service (JMS)* binding when these factors are applicable:

- You need to access a non-IBM vendor messaging system.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider an *Message Queue (MQ)* binding when these factors are applicable:

- You need to access a WebSphere MQ messaging system and need to use the MQ native functions.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider an *MQ JMS* binding when these factors are applicable:

- You need to access a WebSphere MQ messaging system but can do so within a JMS context; that is, the JMS subset of functions is sufficient for your application.
- The services are loosely coupled.

- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

SCA bindings

A Service Component Architecture (SCA) binding lets a service communicate with other services in other modules. An import with an SCA binding lets you access a service in another SCA module. An export with an SCA binding lets you offer a service to other modules.

You use WebSphere Integration Developer to generate and configure SCA bindings on imports and exports in SCA modules.

If modules are running on the same server or are deployed in the same cluster, an SCA binding is the easiest and fastest binding to use.

After the module that contains the SCA binding is deployed to the server, you can use the administrative console to view information about the binding or, in the case of an import binding, to change selected properties of the binding.

Web service bindings

A Web service binding is the means of transmitting messages from a Service Component Architecture (SCA) component to a Web service (and vice versa).

Web service bindings overview:

A Web service import binding allows you to call an external Web service from your Service Component Architecture (SCA) components. A Web service export binding allows you to expose your SCA components to clients as Web services.

With a Web service binding, you access external services using interoperable SOAP messages and qualities of service (QoS).

You use Integration Designer to generate and configure Web service bindings on imports and exports in SCA modules. The following types of Web service bindings are available:

- SOAP1.2/HTTP and SOAP1.1/HTTP

  These bindings are based on Java API for XML Web Services (JAX-WS), a Java programming API for creating Web services.

  – Use SOAP1.2/HTTP if your Web service conforms to the SOAP 1.2 specification.

  – Use SOAP1.1/HTTP if your Web service conforms to the SOAP 1.1 specification.

  Important: When you deploy an application with a Web service (JAX-WS) binding, the target server must not have the **Start components as needed** option selected. See “Checking the server configuration” on page 83 for details.

  When you select one of these bindings, you can send attachments with your SOAP messages.

  The Web service bindings work with standard SOAP messages. Using one of the Web service JAX-WS bindings, however, you can customize the way that SOAP messages are parsed or written. For example, you can handle nonstandard elements in SOAP messages or apply additional processing to the SOAP message. When you configure the binding, you specify a custom data handler that performs this processing on the SOAP message.

You can use policy sets with a Web service (JAX-WS) binding. A policy set is a collection of policy types, each of which provides a quality of service (QoS). For example, the WSAddressing policy set provides a transport-neutral way to uniformly address Web services and messages. You use Integration Designer to select the policy set for the binding.

Note: If you want to use a Security Assertion Markup Language (SAML) policy set, you must perform some additional configuration, as described in "Importing SAML policy sets" on page 81.

- SOAP1.1/HTTP

  Use this binding if you want to create Web services that use a SOAP-encoded message based on Java API for XML-based RPC (JAX-RPC).
- SOAP1.1/JMS

  Use this binding to send or receive SOAP messages using a Java Message Service (JMS) destination.

Regardless of the transport (HTTP or JMS) that is used to convey the SOAP message, Web service bindings always handle request/response interactions synchronously. The thread making the invocation on the service provider is blocked until a response is received from the provider. See "Synchronous invocation" for more information about this invocation style.

Important: The following combinations of Web service bindings cannot be used on exports in the same module. If you need to expose components using more than one of these export bindings, you need to have each in a separate module and then connect those modules to your components using the SCA binding:

- SOAP 1.1/JMS and SOAP 1.1/HTTP using JAX-RPC
- SOAP 1.1/HTTP using JAX-RPC and SOAP 1.1/HTTP using JAX-WS
- SOAP 1.1/HTTP using JAX-RPC and SOAP 1.2/HTTP using JAX-WS

After the SCA module that contains the Web service binding is deployed to the server, you can use the administrative console to view information about the binding or to change selected properties of the binding.

Note: Web services allow applications to interoperate by using standard descriptions of services and standard formats for the messages they exchange. For example, the Web service import and export bindings can interoperate with services that are implemented using Web Services Enhancements (WSE) Version 3.5 and Windows Communication Foundation (WCF) Version 3.5 for Microsoft .NET. When interoperating with such services, you must ensure that:

- The Web Services Description Language (WSDL) file that is used to access a Web service export includes a non-empty SOAP action value for each operation in the interface.
- The Web service client sets either the SOAPAction header or the wsa:Action header when sending messages to a Web service export.

SOAP header propagation:

When handling SOAP messages, you might need to access information from certain SOAP headers in messages that are received, ensure that messages with SOAP headers are sent with specific values, or allow SOAP headers to pass across a module.

When you configure a Web service binding in Integration Designer, you can indicate that you want SOAP headers to be propagated.

- When requests are received at an export or responses received at an import, the SOAP header information can be accessed, allowing logic in the module to be based on header values and allowing those headers to be modified.
- When requests are sent from an export or responses sent from an import, SOAP headers can be included in those messages.

The form and presence of the propagated SOAP headers might be affected by policy sets configured on the import or export, as explained in Table 11 on page 77.

To configure the propagation of SOAP headers for an import or export, you select (from the Properties view of Integration Designer) the **Propagate Protocol Header** tab and select the options you require.

WS-Addressing header

The WS-Addressing header can be propagated by the Web service (JAX-WS) binding.

When you propagate the WS-Addressing header, be aware of the following information:

- If you enable propagation for the WS-Addressing header, the header will be propagated into the module in the following circumstances:
  - When requests are received at an export
  - When responses are received at an import
- The WS-Addressing header is not propagated into outbound messages from WebSphere ESB (that is, the header is not propagated when requests are sent from an import or when responses are sent from the export).

WS-Security header

The WS-Security header can be propagated by both the Web service (JAX-WS) binding and the Web service (JAX-RPC) binding.

The Web services WS-Security specification describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

When you propagate the WS-Security header, be aware of the following information:

- If you enable propagation for the WS-Security header, the header will be propagated across the module in the following circumstances:
  - When requests are received at an export
  - When requests are sent from an import
  - When responses are received at an import
- The header will *not*, by default, be propagated when responses are sent from the export. However, if you set the JVM property `WSSECURITY.ECHO.ENABLED` to `true`, the header will be propagated when responses are sent from the export. In this case, if the WS-Security header on the request path is not modified, WS-Security headers might be automatically echoed from requests into responses.

- The exact form of the SOAP message sent from an import for a request or from an export for a response might not exactly match the SOAP message that was originally received. For this reason, any digital signature should be assumed to become invalid. If a digital signature is required in messages that are sent, it must be established using the appropriate security policy set, and WS-Security headers relating to digital signature in received messages should be removed within the module.

To propagate the WS-Security header, you must include the WS-Security schema with the application module. See "Including the WS-Security schema in an application module" on page 78 for the procedure to include the schema.

How headers are propagated

The way that headers are propagated depends on the security policy setting on the import or export binding, as shown in the Table 11 on page 77:

Table 77. How security headers are passed

	Export binding with no security policy	**Export binding with security policy**
Import binding with no security policy	Security headers are passed as-is through the module. They are not decrypted. The headers are sent outbound in the same form in which they were received. The digital signature might become invalid.	Security headers are decrypted and passed though the module with signature verification and authentication. The decrypted headers are sent outbound. The digital signature might become invalid.
Import binding with security policy	Security headers are passed as-is through the module. They are not decrypted. The headers should not be propagated to the import. Otherwise, an error occurs because of duplication.	Security headers are decrypted and passed though the module with signature verification and authentication. The headers should not be propagated to the import. Otherwise, an error occurs because of duplication.

Configure appropriate policy sets on the export and import bindings, because this isolates the service requester from changes to the configuration or QoS requirements of the service provider. Having standard SOAP headers visible in a module can then be used to influence the processing (for example, logging and tracing) in the module. Propagating SOAP headers across a module from a received message to a sent message does mean that the isolation benefits of the module are reduced.

Standard headers, such as WS-Security headers, should not be propagated on a request to an import or response to an export when the import or export has an associated policy set that would normally result in the generation of those headers. Otherwise, an error will occur because of a duplication of the headers. Instead, the headers should be explicitly removed or the import or export binding should be configured to prevent propagation of protocol headers.

Accessing SOAP headers

When a message that contains SOAP headers is received from a Web service import or export, the headers are placed in the headers section of the service message object (SMO). You can access the header information, as described in "Accessing SOAP header information in the SMO".

Including the WS-Security schema in an application module

The following procedure outlines the steps for including the schema in the application module:

- If the computer on which Integration Designer is running has access to the Internet, perform the following steps:
  1. In the Business Integration perspective, select **Dependencies** for your project.
  2. Expand **Predefined Resources** and select either **WS-Security 1.0 schema files** or **WS-Security 1.1 schema files** to import the schema into your module.
  3. Clean and rebuild the project.
- If a computer on which Integration Designer is running does not have Internet access, you can download the schema to a second computer that does have Internet access. You can then copy it to the computer on which Integration Designer is running.
  1. From the computer that has Internet access, download the remote schema:
     a. Click **File** > **Import** > **Business Integration** > **WSDL and XSD**.
     b. Select **Remote WSDL** or **XSD file**.
     c. Import the following schemas:

        http://www.w3.org/2003/05/soap-envelope/

        http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd

        http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd
  2. Copy the schemas to the computer that does not have Internet access.
  3. From the computer that has no Internet access, import the schema:
     a. Click **File** > **Import** > **Business Integration** > **WSDL and XSD**.
     b. Select **Local WSDL** or **XSD file**.
  4. Change the schema locations for oasis-wss-wssecurity_secext-1.1.xsd:
     a. Open the schema at *workplace_location/module_name/* `StandardImportFilesGen/oasis-wss-wssecurity-secext-1.1.xsd`.
     b. Change:

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='http://www.w3.org/2003/05/soap-envelope/'/>
        ```

        to:

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='../w3/_2003/_05/soap_envelope.xsd'/>
        ```
     c. Change:

        ```
        <xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
        schemaLocation='http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd'/>
        ```

        to:

        ```
        <xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
        schemaLocation='../w3/tr/_2002/rec_xmlenc_core_20021210/xenc-schema.xsd'/>
        ```
  5. Change the schema location for oasis-200401-wss-wssecurity-secext-1.0.xsd:

a. Open the schema at *workplace_location/module_name/*`StandardImportFilesGen/oasis-200401-wss-wssecurity-secext-1.0.xsd`.
b. Change:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
```

to:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="../w3/tr/_2002/rec_xmldsig_core_20020212/xmldsig-core-schema.xsd"/>
```

6. Clean and rebuild the project.

Transport header propagation:

When handling SOAP messages, you might need to access information from certain transport headers in messages that are received, ensure that messages with transport headers are sent with specific values, or allow transport headers to pass across a module.

When you configure a Web service binding in Integration Designer, you can indicate that you want transport headers to be propagated.

- When requests are received at an export or responses are received at an import, the transport header information can be accessed, allowing logic in the module to be based on header values and allowing those headers to be modified.
- When responses are sent from an export or requests are sent from an import, transport headers can be included in those messages.

Specifying propagation of headers

To configure the propagation of transport headers for an import or export, perform the following steps:

1. From the Properties view of Integration Designer, select **Binding** > **Propagation**.
2. Set the transport header propagation option that you require.

Note: Transport header propagation is disabled by default and can be deployed only to a Version 7.0.0.3 (or later) runtime environment. Also note that, for Version 7.0.0.3, transport header propagation is limited to HTTP transport headers only.

If you enable propagation of transport headers, the headers will be propagated across a module from received messages and, if you do not explicitly remove the headers, the headers will be used in subsequent invocations in the same thread.

Note: Transport headers cannot be propagated when you are using the Web service (JAX-RPC) binding.

Accessing the header information

When transport header propagation is enabled for received messages, all transport headers (including customer-defined headers) are visible in the service message object (SMO). You can set the headers to different values or create new ones. Note, however, that there is no checking or validation of the values you set, and any improper or incorrect headers might cause Web service runtime problems.

Consider the following information about the setting of HTTP headers:

- Any changes to the headers that are reserved for the Web service engine will not be honored in the outbound message. For example, the HTTP version or method, Content-Type, Content-Length and SOAPAction headers are reserved for the Web service engine.
- If the header value is a number, the number (rather than the string) should be set directly. For example, use `Max-Forwards = 5` (rather than `Max-Forwards = Max-Forwards: 5`) and `Age = 300` (rather than `Age = Age: 300`).
- If the request message is less than 32 KB in size, the Web service engine removes the Transfer-Encoding header and instead sets the Content-Length header to the fixed size of the message.
- The Content-language is reset by WAS.channel.http on the response path.
- An invalid setting for Upgrade results in a 500 error.
- The following headers append the value reserved by the Web service engine to the customer settings:
  - User-Agent
  - Cache-Control
  - Pragma
  - Accept
  - Connection

You can access the header information in one of the following ways:

- Using a mediation primitive to access the SMO structures

  See the "Related Information" links to find information about using mediation primitives.
- Using the context service SPI

  The following sample code reads the HTTP transport headers from the context service:

```
HeadersType headerType = ContextService.INSTANCE.getHeaders();
HTTPHeaderType httpHeaderType = headerType.getHTTPHeader();
List HTTPHeader  httpHeaders = httpHeaderType.getHeader();
if(httpHeaders!=null){
 for(HTTPHeader httpHeader: httpHeaders){
   String httpHeadername = httpHeader.getName();
   String httpHeaderValue = httpHeader.getValue();
     }
}
List PropertyType  properties = headerType.getProperties();
if(properties!=null){
  for(PropertyType property: properties){
   String propertyName = property.getName();
   String propertyValue = property.getValue().toString();
     }
}
```

Troubleshooting

If you encounter problems when sending the revised headers, you can intercept the TCP/IP message by using tools such as the TCP/IP Monitor in Integration Designer. You access the TCP/IP Monitor by selecting **Run/Debug** > **TCP/IP Monitor** from the Preferences page.

You can also view the header values using the JAX-WS engine trace:
`org.apache.axis2.*=all: com.ibm.ws.websvcs.*=all:`

Working with Web service (JAX-WS) bindings:

When you use Web service (JAX-WS) bindings with your applications, you can add a Security Assertion Markup Language (SAML) quality of service (QOS) to the binding. You must first use the administrative console to import the policy set. You can also use the administrative console to make sure that the server is properly configured for use with the Web service (JAX-WS) binding.

Importing SAML policy sets:

Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information. When you configure a Web service (JAX-WS) binding in Integration Designer, you can specify an SAML policy set. You first use the administrative console of WebSphere ESB to make the SAML policy sets available so that they can be imported into Integration Designer.

Before you begin

The SAML policy sets are typically located in the profile configuration directory:

```
profile_root/config/templates/PolicySets
```

Before you begin this procedure, verify that the following directories (which contain the policy sets) are located in the profile configuration directory:

- `SAML11 Bearer WSHTTPS default`
- `SAML20 Bearer WSHTTPS default`
- `SAML11 Bearer WSSecurity default`
- `SAML20 Bearer WSSecurity default`
- `SAML11 HoK Public WSSecurity default`
- `SAML20 HoK Public WSSecurity default`
- `SAML11 HoK Symmetric WSSecurity default`
- `SAML20 HoK Symmetric WSSecurity default`
- `Username WSHTTPS default`

If the directories are not in the profile configuration directory, copy them to that directory from the following location:

```
app_server_root/profileTemplates/default/documents/config/templates/
PolicySets
```

About this task

You import the policy sets into the administrative console, select the ones you want to make available to Integration Designer, and then save a zip file for each of those policy sets to a location that is accessible by Integration Designer.

Procedure

1. Import the policy sets by following these steps:
   a. From the administrative console, click **Services** > **Policy Sets** > **Application policy sets**.
   b. Click **Import** > **From Default Repository**.
   c. Select the SAML default policy sets, and click **OK**.
2. Export the policy sets so that they can be used by Integration Designer:

a. From the Application policy sets page, select the SAML policy set you want to export, and click **Export**.

   Note: If the Application policy sets page is not currently displayed, click **Services** > **Policy Sets** > **Application Policy Sets** from the administrative console.

b. On the next page, click the .zip file link for the policy set.
c. In the File Download window, click **Save** and indicate a location that is accessible by Integration Designer.
d. Click **Back**.
e. Complete steps 2a on page 82 through 2d on page 82 for each policy set you want to export.

Results

The SAML policy sets are saved in .zip files and are ready to be imported into Integration Designer.

What to do next

Import the policy sets into Integration Designer, as described in the topic "Policy sets".

Invoking web services that require HTTP basic authentication:

HTTP basic authentication employs a user name and password to authenticate a service client to a secure endpoint. You can set up HTTP basic authentication when sending or receiving web service requests.

About this task

You set up HTTP basic authentication for receiving web service requests by configuring the Java API for XML Web Services (JAX-WS) export binding, as described in the "Creating and assigning security roles to web service exports" topic.

HTTP basic authentication can be enabled for web service requests that are sent by a JAX-WS import binding in one of two ways:

- When configuring the import binding in an SCA module, you can select the supplied HTTP authentication policy set named BPMHTTPBasicAuthentication (which is provided with the web service (JAX-WS) import binding) or any other policy set that includes the HTTPTransport policy.
- When constructing the SCA module, you can use mediation flow capabilities to dynamically create a new HTTP authentication header and specify the user name and password information in the header.

Note: The policy set has precedence over the value specified in the header. If you want to use the value set in the HTTP authentication header at run time, do not attach a policy set that includes the HTTPTransport policy. Specifically, do not use the default BPMHTTPBasicAuthentication policy set, and, if you have defined a policy set, make sure it excludes the HTTPTransport policy.

You can find more information about web service policy sets and policy bindings and how they are used in the "Web services policy sets" topic of the WebSphere Application Server Information Center. The link to the topic is listed under "Related information".

Procedure

- To use the supplied policy set, perform the following steps:
  1. Optional: In the administrative console, create a client general policy binding or edit an existing one that includes the HTTPTransport policy with the desired user ID and password values.
  2. In IBM Integration Designer, generate a web service (JAX-WS) import binding and attach the BPMHTTPBasicAuthentication policy set.
  3. Perform *one* of the following steps:
     - In IBM Integration Designer, in the web service (JAX-WS) import binding properties, specify the name of an existing client general policy binding that includes the HTTPTransport policy.
     - After deploying the SCA module, from the administrative console of the process server, select an existing client policy binding or create a new one and associate it with the import binding.
  4. Optional: In the administrative console of the process server, edit the selected policy set binding to specify the required ID and password.
- To specify the user name and password in the HTTP authentication header, perform one of the following sets of steps:
  - Use the HTTP Header Setter mediation primitive in IBM Integration Designer to create the HTTP authentication header, and specify the user name and password.
  - If additional logic is required, use Java code in a custom mediation primitive (as shown in the following example) to:
    1. Create an HTTP authentication header.
    2. Specify the user name and password information.
    3. Add the new HTTP authentication header to HTTPControl.
    4. Set the updated HTTPControl back in the Context service.

```
//Get the HeaderInfoType from contextService
ContextService contextService = (ContextService) ServiceManager.INSTANCE
.locateService("com/ibm/bpm/context/ContextService");
HeaderInfoType headers = contextService.getHeaderInfo();
if(headers == null){
    headers = ContextObjectFactory.eINSTANCE.createHeaderInfoType();
}
//Get the HTTP header and HTTP Control from HeaderInfoType
HTTPHeaderType httpHeaderType = headers.getHTTPHeader();
HTTPControl cp = httpHeaderType.getControl();
HeadersFactory factory = HeadersFactory.eINSTANCE;
if(cp == null){
    cp = factory.createHTTPControl();
}
//Create new HTTPAuthentication and set the HTTPCredentials
HTTPAuthentication authorization = factory.createHTTPAuthentication();
HTTPCredentials credentials = factory.createHTTPCredentials();
authorization.setAuthenticationType(HTTPAuthenticationType.BASIC_LITERAL);
credentials.setUserId("USERNAME");
credentials.setPassword("PASSWORD");
authorization.setCredentials(credentials);
cp.setAuthentication(authorization);
httpHeaderType.setControl(cp);
// Set header info back to the current execution context.
contextService.setHeaderInfo(headers);
```

Checking the server configuration:

When you deploy an application with a Web service (JAX-WS) binding, you must make sure that the server on which the application is deployed does not have the **Start components as needed** option selected.

About this task

You can check to see whether this option is selected by performing the following steps from the administrative console:

Procedure

1. Click **Servers** > **Server types** > **WebSphere application servers**.
2. Click the server name.
3. From the Configuration tab, determine whether **Start components as needed** is selected.
4. Perform one of the following steps:
   - If **Start components as needed** is selected, remove the check, and then click **Apply**.
   - If **Start components as needed** is not selected, click **Cancel**.

Attachments in SOAP messages:

You can send and receive SOAP messages that include binary data (such as PDF files or JPEG images) as attachments. Attachments can be *referenced* (that is, represented explicitly as message parts in the service interface) or *unreferenced* (in which arbitrary numbers and types of attachments can be included).

A referenced attachment can be represented in one of the following ways:

- MTOM attachments use the SOAP Message Transmission Optimization Mechanism (http://www.w3.org/TR/soap12-mtom/) specified encoding. MTOM attachments are enabled through a configuration option in the import and export bindings and are the recommended way to encode attachments for new applications.
- As a wsi:swaRef-typed element in the message schema

  Attachments defined using the wsi:swaRef type conform to the Web Services Interoperability Organization (WS-I) *Attachments Profile Version 1.0* (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), which defines how message elements are related to MIME parts.
- As a top-level message part, using a binary schema type

  Attachments represented as top-level message parts conform to the *SOAP Messages with Attachments* (http://www.w3.org/TR/SOAP-attachments) specification.

  Attachments represented as top-level message parts can also be configured to ensure that the WSDL document and messages produced by the binding conform to the WS-I *Attachments Profile Version 1.0* and the WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html).

An unreferenced attachment is carried in a SOAP message without any representation in the message schema.

In all cases, except MTOM attachments, the WSDL SOAP binding should include a MIME binding for attachments to be used, and the maximum size of the attachments should not exceed 20 MB.

Note: To send or receive SOAP messages with attachments, you must use one of the Web service bindings based on the Java API for XML Web Services (JAX-WS).

How to choose the appropriate attachment style:

When designing a new service interface that includes binary data, consider how that binary data is carried in the SOAP messages that are sent and received by the service.

Message Transmission Optimization Mechanism (MTOM) should be used for attachments if the connected web service application supports it. If not, the following diagram shows how other attachment styles are chosen. Use the following set of questions to determine the appropriate attachment style:

MTOM attachments: top-level message parts:

You can send and receive web service messages which include SOAP Message Transmission Optimization Mechanism (MTOM) attachments. In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachment or attachments are in subsequent parts.

By sending or receiving a referenced attachment in a SOAP message, the binary data that makes up the attachment (which is often quite large) is held separately

from the SOAP message body so that it does not need to be parsed as XML. This results in more efficient processing than if the binary data were held within an XML element.

The following is a sample of an MTOM SOAP message:

```
... other transport headers ...
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812; type="application/xop+xml"; start="
<0.urn:uuid:0FE43E4D025F0BF3DC115824676468130apache.org>"; start-info="text/xml"; charset=UTF-8

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
   <0.urn:uuid:0FE43E4D025F0BF3DC115824676468130apache.org>

<?xml version="1.0" encoding="UTF-8"?>
         <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
            <soapenv:Header/>
            <soapenv:Body>
               <sendImage xmlns="http://org/apache/axis2/jaxws/sample/mtom">
                  <input>
                     <imageData><xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:1.urn:uuid:0FE43E4D025F0BF3DC115824676468110apache.org"/></imageData>
                  </input>
               </sendImage>
            </soapenv:Body>
         </soapenv:Envelope>
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: text/plain
content-transfer-encoding: binary
content-id:
         <1.urn:uuid:0FE43E4D025F0BF3DC115824676468110apache.org>

... binary data goes here ...
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812--
```

Note that in the MTOM sample, the content-type for the SOAP envelope is `application/xop+xml` and the binary data is replaced by an `xop:Include` element like below:

```
<xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:1.urn:uuid:0FE43E4D025F0BF3DC115824676468110apache.org"/>
```

Inbound processing of referenced attachments

When a client passes a SOAP message with an attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the attachment binary in the business object.

Figure 279. How the Web service (JAX-WS) export binding processes a SOAP message with a referenced attachment

MTOM attachment attributes

- MTOM can support attachment elements within nested structures.
- MTOM is only available for the base64Binary type.
- MTOM can support attachment elements within nested structures meaning that the `bodyPath` for MTOM attachments are the `xpath` location for the element where the MTOM attachment is held. The computing logic for `bodyPath` is strictly following the schema to generate the `xpath` location as shown in the examples below:
  - For a non-array type (`maxOccurs` is 1): /sendImage/input/imageData

– For a array type (`maxOccurs` > 1): /sendImage/input/imageData[1]

- Mixed attachment types are not supported, meaning that if MTOM is enabled on the import binding, the MTOM attachment will be generated. If MTOM is disabled or if the MTOM configuration value is left as the default value on the export binding, the incoming MTOM message is not supported.

Referenced attachments: swaRef-typed elements:

You can send and receive SOAP messages that include attachments represented in the service interface as swaRef-typed elements.

An swaRef-typed element is defined in the Web Services Interoperability Organization (WS-I) *Attachments Profile* Version 1.0 (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), which defines how message elements are related to MIME parts.

In the SOAP message, the SOAP body contains an swaRef-typed element that identifies the content ID of the attachment.

Figure 280. A SOAP message with an swaRef element

The WSDL for this SOAP message contains an swaRef-typed element within a message part that identifies the attachment.

```
<element name="sendPhoto">
  <complexType>
     <sequence>
       <element name="Photo" type="wsi:swaRef"/>
     </sequence>
  </complexType>
 </element>
```

The WSDL should also contain a MIME binding that indicates MIME multipart messages are to be used.

Note: The WSDL does *not* include a MIME binding for the specific swaRef-typed message element, because MIME bindings apply only to top-level message parts.

Attachments represented as swaRef-typed elements can be propagated only across mediation flow components. If an attachment must be accessed by or propagated to another component type, use a mediation flow component to move the attachment to a location that is accessible by that component.

Inbound processing of attachments

You use Integration Designer to configure an export binding to receive the attachment. You create a module and its associated interface and operations, including an element of type swaRef. You then create a Web service (JAX-WS) binding.

Note: See the "Working with attachments" topic in the Integration Designer information center for more detailed information.

When a client passes a SOAP message with an swaRef attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the content ID of the attachment in the business object.

Figure 281. How the Web service (JAX-WS) export binding processes a SOAP message with an swaRef attachment

Accessing attachment metadata in a mediation flow component

As shown in Figure 28 on page 89, when swaRef attachments are accessed by components, the attachment content identifier appears as an element of type swaRef.

Each attachment of a SOAP message also has a corresponding `attachments` element in the SMO. When using the WS-I swaRef type, the `attachments` element includes the attachment content type and content ID as well as the actual binary data of the attachment.

To obtain the value of an swaRef attachment, it is therefore necessary to obtain the value of the swaRef-typed element, and then locate the `attachments` element with the corresponding `contentID` value. Note that the `contentID` value typically has the `cid:` prefix removed from the swaRef value.

Figure 282. How swaRef attachments appear in the SMO

Outbound processing

You use Integration Designer to configure a Web service (JAX-WS) import binding to invoke an external Web service. The import binding is configured with a WSDL document that describes the Web service to be invoked and defines the attachment that will be passed to the Web service.

When an SCA message is received by a Web service (JAX-WS) import binding, swaRef-typed elements are sent as attachments if the import is wired to a mediation flow component and the swaRef-typed element has a corresponding `attachments` element.

For outbound processing, swaRef-typed elements are always sent with their content ID values; however the mediation module must ensure that there is a corresponding `attachments` element with a matching `contentID` value.

Note: To conform to the WS-I Attachments Profile, the `content ID` value should follow the "content-id part encoding," as described in section 3.8 of the WS-I *Attachments Profile* 1.0.

Figure 283. How the Web service (JAX-WS) import binding generates a SOAP message with an swaRef attachment

Setting attachment metadata in a mediation flow component

If, in the SMO, there is an swaRef-typed element value and an `attachments` element, the binding prepares the SOAP message (with the attachment) and sends it to a recipient.

Figure 284. How an swaRef attachment in the SMO is accessed to create the SOAP message

The `attachments` element is present in the SMO only if a mediation flow component is connected directly to the import or export; it does not get passed across other component types. If the values are needed in a module that contains other component types, a mediation flow component should be used to copy the

values into a location where they can then be accessed in the module, and another mediation flow component used to set the correct values before an outbound invocation by way of a Web service import.

Important: As described in "XML representation of SMO," the XSL Transformation mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The XSL Transformation mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Referenced attachments: top-level message parts:

You can send and receive SOAP messages that include binary attachments that are declared as parts in your service interface.

In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachment or attachments are in subsequent parts.

What is the advantage of sending or receiving a referenced attachment in a SOAP message? The binary data that makes up the attachment (which is often quite large) is held separately from the SOAP message body so that it does not need to be parsed as XML. This results in more efficient processing than if the binary data were held within an XML element.

Types of SOAP messages with referenced attachments

Beginning with Version 7.0.0.3 of WebSphere ESB, you have a choice about how the SOAP message is generated:

- **WS-I-compliant messages**

  The runtime can generate SOAP messages that comply with the WS-I *Attachments Profile Version 1.0* and the WS-I *Basic Profile Version 1.1*. In a SOAP message that is compliant with these profiles, only one message part is bound to the SOAP body; for those that are bound as attachments, the content-id part encoding (as described in the WS-I *Attachments Profile Version 1.0*) is used to relate the attachment to the message part.
- **Non-WS-I-compliant messages**

  The runtime can generate SOAP messages that do not comply with the WS-I profiles but that are compatible with the messages generated in Version 7.0 or 7.0.0.2 of WebSphere ESB. The SOAP messages use top-level elements named after the message part with an `href` attribute that holds the attachment `content-id`, but the content-id part encoding (as described in the WS-I *Attachments Profile Version 1.0*) is not used.

Selecting WS-I compliance for Web service exports

You use Integration Designer to configure an export binding. You create a module and its associated interface and operations. You then create a Web service (JAX-WS) binding. The Referenced attachments page displays all the binary parts

from the created operation, and you select which parts will be attachments. You then specify, on the Specify the WS-I AP 1.0 compliance page of Integration Designer, one of the following choices:

- **Use WS-I AP 1.0 compliant SOAP message**

  If you select this option, you also specify which message part should be bound to the SOAP body.

  Note: This option can be used only when the corresponding WSDL file is also WS-I compliant.

  A WSDL file that is generated by Integration Designer Version 7.0.0.3 is compliant with WS-I. However, if you import a WSDL file that is not compliant with WS-I, you cannot select this option.
- **Use non WS-I AP 1.0 compliant SOAP message**

  If you select this option, which is the default, the first message part is bound to the SOAP body.

Note: Only top-level message parts (that is, elements defined in the WSDL portType as parts within the input or output message) that have a binary type (either base64Binary or hexBinary) can be sent or received as referenced attachments.
See the "Working with attachments" topic in the Integration Designer information center for more detailed information.

For WS-I-compliant messages, the content-ID that is generated in the SOAP message is a concatenation of the following elements:

- The value of the `name` attribute of the `wsdl:part` element referenced by the `mime:content`
- The character `=`
- A globally unique value, such as a UUID
- The character `@`
- A valid domain name

Inbound processing of referenced attachments

When a client passes a SOAP message with an attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the attachment binary in the business object.

Figure 285. How the Web service (JAX-WS) export binding processes a WS-I compliant SOAP message with a referenced attachment

Accessing attachment metadata in a mediation flow component

As shown in Figure 31 on page 93, when referenced attachments are accessed by components, the attachment data appears as a byte array.

Each referenced attachment of a SOAP message also has a corresponding `attachments` element in the SMO. The `attachments` element includes the attachment content type and the path to the message body element where the attachment is held.

Figure 286. How referenced attachments appear in the SMO

Important: The path to the message body element is not automatically updated if the message is transformed and the attachment moved. You can use a mediation flow to update the `attachments` element with the new path (for example, as part of the transform or using a separate message element setter).

How outbound SOAP messages are constructed

You use Integration Designer to configure a Web service (JAX-WS) import binding to invoke an external Web service. The import binding is configured with a WSDL document that describes the Web service to be invoked and defines which message parts should be passed as attachments. You can also indicate, on the Specify the WS-I AP 1.0 compliance page of Integration Designer, one of the following choices:

- **Use WS-I AP 1.0 compliant SOAP message**

  If you select this option, you also specify which message part should be bound to the SOAP body; all others are bound to attachments or headers. Messages sent by the binding do not include elements in the SOAP body that refer to the attachments; the relationship is expressed by way of the attachment content ID including the message part name.
- **Use non WS-I AP 1.0 compliant SOAP message**

  If you select this option, which is the default, the first message part is bound to the SOAP body; all others are bound to attachments or headers. Messages sent by the binding include one or more elements in the SOAP body that refer to the attachments by way of an `href` attribute.

Note: The part that represents an attachment, as defined in the WSDL, must be a simple type (either base64Binary or hexBinary). If a part is defined by a complexType, that part cannot be bound as an attachment.

Outbound processing of referenced attachments

The import binding uses information in the SMO to determine how the binary top-level message parts are sent as attachments.

Figure 287. How the referenced attachment in the SMO is accessed to create the SOAP message

The `attachments` element is present in the SMO only if a mediation flow component is connected directly to the import or export; it does not get passed across other component types. If the values are needed in a module that contains other component types, a mediation flow component should be used to copy the values into a location where they can then be accessed in the module, and another mediation flow component used to set the correct values before an outbound invocation by way of a Web service import.

The binding uses a combination of the following conditions to determine how (or whether) the message is sent:

- Whether there is a WSDL MIME binding for the top-level binary message part and, if so, how the content type is defined
- Whether there is an `attachments` element in the SMO whose `bodyPath` value references a top-level binary part

How attachments are created when an `attachment` element exists in the SMO

The following table shows how an attachment is created and sent if the SMO contains an `attachment` element with a `bodyPath` that matches a message name part:

Table 78. How the attachment is generated

Status of WSDL MIME binding for top-level binary message part	How message is created and sent
Present with one of the following: • No defined content type for the message part • Multiple content types defined • Wildcard content type defined	Message part is sent as an attachment. `Content-Id` is set to the value in the attachments element if present; otherwise, one is generated. `Content-Type` is set to the value in the attachments element if present; otherwise, it is set to `application/octet-stream`.
Present with single, non-wildcard content for the message part	Message part is sent as an attachment. `Content-Id` is set to the value in the attachments element if present; otherwise, one is generated. `Content-Type` is set to the value in the attachments element if present; otherwise, it is set to the type defined in the WSDL MIME content element.
Not present	Message part is sent as an attachment. `Content-Id` is set to the value in the attachments element if present; otherwise, one is generated. `Content-Type` is set to the value in the attachments element if present; otherwise, it is set to `application/octet-stream`. **Note:** Sending message parts as attachments when not defined as such in the WSDL may break compliance with the WS-I Attachments Profile 1.0 and so should be avoided if possible.

How attachments are created when no `attachment` element exists in the SMO

The following table shows how an attachment is created and sent if the SMO does not contain an `attachment` element with a `bodyPath` that matches a message name part:

Table 79. How the attachment is generated

Status of WSDL MIME binding for top-level binary message part	How message is created and sent
Present with one of the following: • No defined content type for the message part • Multiple content types defined • Wildcard content type defined	Message part is sent as an attachment. `Content-Id` is generated. `Content-Type` is set to `application/octet-stream`.
Present with single, non-wildcard content for the message part	Message part is sent as an attachment. `Content-Id` is generated. `Content-Type` is set to the type defined in the WSDL MIME content element.
Not present	Message part is not sent as an attachment.

Important: As described in "XML representation of SMO," the XSL Transformation mediation primitive transforms messages using an XSLT 1.0 transformation. The

transformation operates on an XML serialization of the SMO. The XSL Transformation mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Unreferenced attachments:

You can send and receive *unreferenced* attachments that are not declared in the service interface.

In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachments are in subsequent parts. No reference to the attachment is included in the SOAP body.

Figure 288. A SOAP message with an unreferenced attachment

You can send a SOAP message with an unreferenced attachment through a Web service export to a Web service import. The output message, which is sent to the target Web service, contains the attachment.

Figure 289. An attachment passing through an SCA module

In Figure 35 on page 97, the SOAP message, with the attachment, passes through without modification.

You can also modify the SOAP message by using a mediation flow component. For example, you can use the mediation flow component to extract data from the SOAP message (binary data in the body of the message, in this case) and create a SOAP with attachments message. The data is processed as part of the attachments element of a service message object (SMO).

Figure 290. A message processed by a mediation flow component

Conversely, the mediation flow component can transform the incoming message by extracting and encoding the attachment and then transmitting the message with no attachments.

Instead of extracting data from an incoming SOAP message to form a SOAP with attachments message, you can obtain the attachment data from an external source, such as a database.

Figure 291. An attachment obtained from a database and added to the SOAP message

Conversely, the mediation flow component can extract the attachment from an incoming SOAP message and process the message (for example, store the attachment in a database).

Unreferenced attachments can be propagated only across mediation flow components. If an attachment must be accessed by or propagated to another component type, use a mediation flow component to move the attachment to a location that is accessible by that component.

Important: As described in "XML representation of SMO," the XSL Transformation mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The XSL Transformation mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Use of WSDL document style binding with multipart messages:

The Web Services Interoperability Organization (WS-I) organization has defined a set of rules regarding how Web services should be described by way of a WSDL and how the corresponding SOAP messages should be formed, in order to ensure interoperability.

These rules are specified in the WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html). In particular, the WS-I Basic Profile 1.1 R2712 states: "A document-literal binding MUST be serialized as an ENVELOPE with a

soap:Body whose child element is an instance of the global element declaration referenced by the corresponding wsdl:message part.”

This means that, when using a document style SOAP binding for an operation with messages (input, output, or fault) that are defined with multiple parts, only one of those parts should be bound to the SOAP body in order to be compliant with the WS-I Basic Profile 1.1.

Further, the WS-I Attachments Profile 1.0 R2941 states: “A wsdl:binding in a DESCRIPTION SHOULD bind every wsdl:part of a wsdl:message in the wsdl:portType to which it refers to one of soapbind:body, soapbind:header, soapbind:fault , soapbind:headerfault, or mime:content.”.

This means that, when using a document style SOAP binding for an operation with messages (input, output, or fault) that are defined with multiple parts, all parts other than the one selected to be bound to the SOAP body must be bound as attachments or headers.

The following approach is used when WSDL descriptions are generated for exports with Web service (JAX-WS and JAX-RPC) bindings in this case:

- You can choose which message part is bound to the SOAP body if there is more than one non-binary-typed element. If there is a single non-binary typed element, that element is automatically bound to the SOAP body.
- For the JAX-WS binding, all other message parts of type "hexBinary" or "base64Binary" are bound as referenced attachments. See “Referenced attachments: top-level message parts” on page 91.
- All other message parts are bound as SOAP headers.

The JAX-RPC and JAX-WS import bindings honor the SOAP binding in an existing WSDL document with multipart document style messages even if it does bind multiple parts to the SOAP body; however, you are not able to generate Web service clients for such WSDL documents in Rational Application Developer.

Note: The JAX-RPC binding does not support attachments.

The recommended pattern when using multipart messages with an operation that has document style SOAP binding is therefore:

1. Use the document/literal wrapped style. In this case, messages always have a single part; however, attachments have to be unreferenced (as described in “Unreferenced attachments” on page 97) or swaRef-typed (as described in “Referenced attachments: swaRef-typed elements” on page 87) in this case.
2. Use the RPC/literal style. In this case, there are no restrictions on the WSDL binding in terms of number of parts bound to the SOAP body; the SOAP message that results always has a single child that represents the operation being invoked, with the message parts being children of that element.
3. For the JAX-WS binding, have at most one message part that is not of type "hexBinary" or "base64Binary", unless it is acceptable to bind the other non-binary parts to SOAP headers.
4. Any other cases are subject to the behavior described.

Note: Additional restrictions exist when you use are using SOAP messages that do not conform to the WS-I *Basic Profile Version 1.1*.

- The first message part should be non-binary.

- When receiving multipart document-style SOAP messages with referenced attachments, the JAX-WS binding expects each referenced attachment to be represented by a SOAP body child element with an href attribute value that identifies the attachment by its content ID. The JAX-WS binding sends referenced attachments for such messages in the same way. This behavior is not compliant with the WS-I Basic Profile.

  To ensure that your messages comply with the Basic Profile, follow approach 1 on page 100 or 2 on page 100 in the previous list or avoid the use of referenced attachments for such messages and use unreferenced or swaRef-typed attachments instead.

HTTP bindings

The HTTP binding is designed to provide Service Component Architecture (SCA) connectivity to HTTP. Consequently, existing or newly-developed HTTP applications can participate in Service Oriented Architecture (SOA) environments.

Hypertext Transfer Protocol (HTTP) is a widely-used protocol for transferring information on the Web. When you are working with an external application that uses the HTTP protocol, an HTTP binding is necessary. The HTTP binding handles the transformation of data passed in as a message in a native format to a business object in an SCA application. The HTTP binding also can transform data passed out as a business object to the native format expected by the external application. for an incoming messaging.

Note: If you want to interact with clients and services that use the Web services SOAP/HTTP protocol, consider using one of the Web service bindings, which provide additional functionality with respect to handling Web services standard qualities of service.

Some common scenarios for using the HTTP binding are described in the following list:

- SCA-hosted services can invoke HTTP applications using an HTTP import.
- SCA-hosted services can expose themselves as HTTP-enabled applications, so they can be used by HTTP clients, using an HTTP export.
- WebSphere ESB and Process Server can communicate between themselves across an HTTP infrastructure, consequently users can manage their communications according to corporate standards.
- WebSphere ESB and Process Server can act as mediators of HTTP communications, transforming and routing messages, which improves the integration of applications using a HTTP network.
- WebSphere ESB and Process Server can be used to bridge between HTTP and other protocols, such as SOAP/HTTP Web services, Java Connector Architecture (JCA)-based resource adapters, JMS, and so on.

Detailed information about creating HTTP import and export bindings can be found in the Integration Designer information center. See the **Developing integration applications** > **Accessing external services with HTTP**> topics.

HTTP bindings overview:

The HTTP binding provides connectivity to HTTP-hosted applications. It mediates communication between HTTP applications and allows existing HTTP-based applications to be called from a module.

HTTP import bindings

The HTTP import binding provides outbound connectivity from Service Component Architecture (SCA) applications to an HTTP server or applications.

The import invokes an HTTP endpoint URL. The URL can be specified in one of three ways:

- The URL can be set dynamically in the HTTP headers by way of the dynamic override URL.
- The URL can be set dynamically in the SMO target address element.
- The URL can be specified as a configuration property on the import.

This invocation is always synchronous in nature.

Although HTTP invocations are always request-reply, the HTTP import supports both one-way and two-way operations and ignores the response in the case of a one-way operation.

HTTP export bindings

The HTTP export binding provides inbound connectivity from HTTP applications to an SCA application.

A URL is defined on the HTTP export. HTTP applications that want to send request messages to the export use this URL to invoke the export.

The HTTP export also supports pings.

HTTP bindings at runtime

An import with an HTTP binding at runtime sends a request with or without data in the body of the message from the SCA application to the external Web service. The request is made from the SCA application to the external Web service, as shown in Figure 38 on page 102.

Figure 292. Flow of a request from the SCA application to the Web application

Optionally, the import with the HTTP binding can receive data back from the Web application in a response to the request.

With an export, the request is made by a client application to a Web service, as shown in Figure 39 on page 102.

Figure 293. Flow of a request from the Web service to the client application.

The Web service is a Web application running on the server. The export is implemented in that Web application as a servlet so the client sends its request to a URL address. The servlet passes the request to the SCA application in the runtime.

Optionally, the export may send data to the client application in response to the request.

HTTP headers:

HTTP import and export bindings allow configuration of HTTP headers and their values to be used for outbound messages. The HTTP import uses these headers for requests, and the HTTP export uses them for responses.

Statically configured headers and control information take precedence over values dynamically set at runtime. However, the dynamic override URL, Version, and Method control values override the static values, which are otherwise considered defaults.

The binding supports the dynamic nature of the HTTP import URL by determining the value of HTTP target URL, Version, and Method at run time. These values are determined by extracting the value of Endpoint Reference, Dynamic Override URL, Version, and Method.

- For Endpoint Reference, use com.ibm.websphere.sca.addressing.EndpointReference APIs or set the /headers/SMOHeader/Target/address field in the SMO header.
- For Dynamic Override URL, Version, and Method, use the HTTP control parameters section of the Service Component Architecture (SCA) message. Note that the Dynamic Override URL takes precedence over the target Endpoint Reference; however, the Endpoint Reference applies across bindings, so it is the preferred approach and should be used where possible.

The control and header information for outbound messages under HTTP export and import bindings is processed in the following order:

1. Header and control information excluding HTTP dynamic override URL, Version, and Method from the SCA Message (lowest priority)
2. Changes from the administrative console on the export/import level
3. Changes from the administrative console on the method level of the export or import
4. Target address specified by way of the Endpoint Reference or the SMO header
5. Dynamic Override URL, Version, and Method from the SCA message
6. Headers and control information from the data handler or data binding (highest priority)

The HTTP export and import will populate inbound direction headers and control parameters with data from the incoming message (HTTPExportRequest and HTTPImportResponse) only if protocol header propagation is set to `True`. Inversely, the HTTP export and import will read and process outbound headers and control parameters (HTTPExportResponse and HTTPImportRequest) only if protocol header propagation is set to `True`.

Note: Data handler or data binding changes to headers or control parameters in the import response or export request will not alter the processing instructions of the message inside the import or export binding and should be used only to propagate modified values to downstream SCA components.

The context service is responsible for propagating the context (including the protocol headers, such as the HTTP header, and the user context, such as account ID) along an SCA invocation path. During development in IBM Integration Designer, you can control the propagation of context by way of import and export properties. For more details, see the import and export bindings information in the IBM Integration Designer information center.

Supplied HTTP header structures and support

Table 14 on page 104 itemizes the request/response parameters for HTTP Import and HTTP Export requests and responses.

Table 80. Supplied HTTP header information

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
URL	Ignored	Not set	Read from the request message. **Note:** Query string is also part of the URL control parameter.	Ignored
Version (possible values: 1.0, 1.1; default is 1.1)	Ignored	Not set	Read from the request message	Ignored
Method	Ignored	Not set	Read from the request message	Ignored
Dynamic Override URL	If set in the data handler or data binding, overrides the HTTP Import URL. Written to the message in the request line. **Note:** Query string is also part of the URL control parameter.	Not set	Not set	Ignored
Dynamic Override Version	If set, overrides the HTTP Import Version. Written to the message in the request line.	Not set	Not set	Ignored
Dynamic Override Method	If set, overrides the HTTP Import Method. Written to the message in the request line.	Not set	Not set	Ignored
Media Type (This control parameter carries part of the value of the Content-Type HTTP header.)	If present, written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data handler or data binding.	Read from the response message, Content-Type header	Read from the request message, Content-Type header	If present, written to the message as part of Content-Type header. **Note:** This control element value should be provided by the data handler or data binding.

Table 80. Supplied HTTP header information (continued)

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
Character set (default: UTF-8)	If present, written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data binding.	Read from the response message, Content-Type header	Read from the request message, Content-Type header	Supported; written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data binding.
Transfer Encoding (Possible values: chunked, identity; default is identity)	If present, written to the message as a header and controls how the message transformation is encoded.	Read from the response message	Read from the request message	If present, written to the message as a header and controls how the message transformation is encoded.
Content Encoding (Possible values: gzip, x-gzip, deflate, identity; default is identity)	If present, written to the message as a header and controls how the payload is encoded.	Read from the response message	Read from the request message	If present, written to the message as a header and controls how the payload is encoded.
Content-Length	Ignored	Read from the response message	Read from the request message	Ignored
StatusCode (default: 200)	Not supported	Read from the response message	Not supported	If present, written to the message in the response line
ReasonPhrase (default: OK)	Not supported	Read from the response message	Not supported	Control value ignored. The message response line value is generated from the StatusCode.
Authentication (contains multiple properties)	If present, used to construct the Basic Authentication header. **Note:** The value for this header will be encoded only on the HTTP protocol. In the SCA, it will be decoded and passed as clear text.	Not applicable	Read from the request message Basic Authentication header. The presence of this header does not indicate the user has been authenticated. Authentication should be controlled in the servlet configuration. **Note:** The value for this header will be encoded only on the HTTP protocol. In the SCA, it will be decoded and passed as clear text.	Not applicable
Proxy (contains multiple properties: Host, Port, Authentication)	If present, used to establish connection through proxy.	Not applicable	Not applicable	Not applicable

Table 80. Supplied HTTP header information (continued)

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
SSL (contains multiple properties: Keystore, Keystore Password, Trustore, Trustore Password, ClientAuth)	If populated and the destination url is HTTPS, it is used to establish a connection through SSL.	Not applicable	Not applicable	Not applicable

HTTP data bindings:

For each different mapping of data between a Service Component Architecture (SCA) message and an HTTP protocol message, a data handler or an HTTP data binding must be configured. Data handlers provide a binding-neutral interface that allows reuse across transport bindings and represent the recommended approach; data bindings are specific to a particular transport binding. HTTP-specific data binding classes are supplied; you can also write custom data handlers or data bindings.

Note: The three HTTP data binding classes described in this topic (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML, and HTTPServiceGatewayDataBinding) are deprecated as of WebSphere ESB Version 7.0. Instead of using the data bindings described in this topic, consider the following data handlers:

- Use SOAPDataHandler instead of HTTPStreamDataBindingSOAP.
- Use UTF8XMLDataHandler instead of HTTPStreamDataBindingXML
- Use GatewayTextDataHandler instead of HTTPServiceGatewayDataBinding

Data bindings are provided for use with HTTP imports and HTTP exports: binary data binding, XML data binding, and SOAP data binding. A response data binding is not required for one-way operations. A data binding is represented by the name of a Java class whose instances can convert both from HTTP to ServiceDataObject and vice-versa. A function selector must be used on an export which, in conjunction with method bindings, can determine which data binding is used and which operation is invoked. The supplied data bindings are:

- Binary data bindings, which treat the body as unstructured binary data. The implementation of the binary data binding XSD schema is as follows:

```
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:tns="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="HTTPBaseBody">
    <xsd:sequence/>
  </xsd:complexType>

  <xsd:complexType name="HTTPBytesBody">
    <xsd:complexContent>
      <xsd:extension base="tns:HTTPBaseBody">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:hexBinary"/>
```

```
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

- XML data bindings, which support the body as XML data. The implementation of the XML data binding is similar to the JMS XML data binding and has no restrictions on the interface schema.
- SOAP data bindings, which support the body as SOAP data. The implementation of the SOAP data binding has no restrictions on the interface schema.

Implementing custom HTTP data bindings

This section describes how to implement a custom HTTP data binding.

Note: The recommended approach is to implement a custom data handler because it can be reused across transport bindings.

HTTPStreamDataBinding is the principal interface for handling custom HTTP messages. The interface is designed to allow handling of large payloads. However, in order for such implementation to work, this data binding must return the control information and headers before writing the message into the stream.

The methods and their order of execution, listed below, must be implemented by the custom data binding.

To customize a data binding, write a class that implements HTTPStreamDataBinding. The data binding should have four private properties:

- private DataObject pDataObject
- private HTTPControl pCtrl
- private HTTPHeaders pHeaders
- private yourNativeDataType nativeData

The HTTP binding will invoke the customized data binding in the following order:

- Outbound processing (DataObject to Native format):
  1. setDataObject(...)
  2. setHeaders(...)
  3. setControlParameters(...)
  4. setBusinessException(...)
  5. convertToNativeData()
  6. getControlParameters()
  7. getHeaders()
  8. write(...)
- Inbound processing (Native format to DataObject):
  1. setControlParameters(...)
  2. setHeaders(...)
  3. convertFromNativeData(...)
  4. isBusinessException()
  5. getDataObject()
  6. getControlParameters()

7. getHeaders()

You need to invoke setDataObject(...) in convertFromNativeData(...) to set the value of dataObject, which is converted from native data to the private property "pDataObject".

```
public void setDataObject(DataObject dataObject)
   throws DataBindingException {
  pDataObject = dataObject;

}
public void setControlParameters(HTTPControl arg0) {
  this.pCtrl = arg0;
}

public void setHeaders(HTTPHeaders arg0) {
  this.pHeaders = arg0;
}
/*
* Add http header "IsBusinessException" in pHeaders.
* Two steps:
* 1.Remove all the header with name IsBusinessException (case-insensitive) first.
*   This is to make sure only one header is present.
* 2.Add the new header "IsBusinessException"
*/
public void setBusinessException(boolean isBusinessException) {
  //remove all the header with name IsBusinessException (case-insensitive) first.
  //This is to make sure only one header is present.
  //add the new header "IsBusinessException", code example:
  HTTPHeader header=HeadersFactory.eINSTANCE.createHTTPHeader();
  header.setName("IsBusinessException");
  header.setValue(Boolean.toString(isBusinessException));
  this.pHeaders.getHeader().add(header);

}
public HTTPControl getControlParameters() {
  return pCtrl;
}
public HTTPHeaders getHeaders() {
  return pHeaders;
}
public DataObject getDataObject() throws DataBindingException {
  return pDataObject;
}
/*
* Get header "IsBusinessException" from pHeaders, return its boolean value
*/
public boolean isBusinessException() {
  String headerValue = getHeaderValue(pHeaders,"IsBusinessException");
  boolean result=Boolean.parseBoolean(headerValue);
  return result;
}
public void convertToNativeData() throws DataBindingException {
  DataObject dataObject = getDataObject();
  this.nativeData=realConvertWorkFromSDOToNativeData(dataObject);
}
public void convertFromNativeData(HTTPInputStream arg0){
  //Customer-developed method to
  //Read data from HTTPInputStream
  //Convert it to DataObject
  DataObject dataobject=realConvertWorkFromNativeDataToSDO(arg0);
  setDataObject(dataobject);
}
```

```
public void write(HTTPOutputStream output) throws IOException {
  if (nativeData != null)
  output.write(nativeData);
}
```

EJB bindings

Enterprise JavaBeans (EJB) import bindings enable Service Component Architecture (SCA) components to invoke services provided by Java EE business logic running on a Java EE server. EJB export bindings allow SCA components to be exposed as Enterprise JavaBeans so that Java EE business logic can invoke SCA components otherwise unavailable to them.

EJB import bindings:

EJB import bindings allow an SCA module to call EJB implementations by specifying the way that the consuming module is bound to the external EJB. Importing services from an external EJB implementation allows users to plug their business logic into the WebSphere ESB environment and participate in a business process.

You use Integration Designer to create EJB import bindings. You can use either of the following procedures to generate the bindings:

- Creating EJB import using the external service wizard

  You can use the external service wizard in Integration Designer to build an EJB import based on an existing implementation. The external service wizard creates services based on criteria that you provide. It then generates business objects, interfaces, and import files based on the services discovered.
- Creating EJB import using the assembly editor

  You can create an EJB import within an assembly diagram using the Integration Designer assembly editor. From the palette, you can use either an Import or use a Java class to create the EJB binding.

The generated import has data bindings that make the Java-WSDL connection instead of requiring a Java bridge component. You can directly wire a component with a Web Services Description Language (WSDL) reference to the EJB import that communicates to an EJB-based service using a Java interface.

The EJB import can interact with Java EE business logic using either the EJB 2.1 programming model or the EJB 3.0 programming model.

The invocation to the Java EE business logic can be local (for EJB 3.0 only) or remote.

- Local invocation is used when you want to call Java EE business logic that resides on the same server as the import.

Figure 294. Local invocation of an EJB (EJB 3.0 only)

- Remote invocation is used when you want to call Java EE business logic that does not reside on the same server as the import.

  For example, in the following figure, an EJB import uses the Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP) to invoke an EJB method on another server.

Figure 295. Remote invocation of an EJB

When it configures the EJB binding, Integration Designer uses the JNDI name to determine the EJB programming model level and the type of invocation (local or remote).

EJB import bindings contain the following major components:

- JAX-WS data handler
- EJB fault selector
- EJB import function selector

If your user scenario is not based on the JAX-WS mapping, you might need a custom data handler, function selector, and fault selector to perform the tasks otherwise completed by the components that are part of the EJB import bindings. This includes the mapping normally completed by the custom mapping algorithm.

EJB export bindings:

External Java EE applications can invoke an SCA component by way of an EJB export binding. Using an EJB export lets you expose SCA components so that external Java EE applications can invoke those components using the EJB programming model.

Note: The EJB export is a stateless bean.

You use Integration Designer to create EJB bindings. You can use either of the following procedures to generate the bindings:

- Creating EJB export bindings using the external service wizard

  You can use the external service wizard in Integration Designer to build an EJB export service based on an existing implementation. The external service wizard creates services based on criteria that you provide. It then generates business objects, interfaces, and export files based on the services discovered.
- Creating EJB export bindings using the assembly editor

  You can create an EJB export using the Integration Designer assembly editor.

Important: A Java 2 Platform, Standard Edition (J2SE) client cannot invoke the EJB export client that is generated in Integration Designer.

You can generate the binding from an existing SCA component, or you can generate an export with an EJB binding for a Java interface.

- When you generate an export for an existing SCA component that has an existing WSDL interface, the export is assigned a Java interface.
- When you generate an export for a Java interface, you can select either a WSDL or a Java interface for the export.

  Note: A Java interface used to create an EJB export has the following limitations with regard to the objects (input and output parameters and exceptions) passed as parameters on a remote call:
  - They must be of concrete type (instead of an interface or abstract type).
  - They must conform to the Enterprise JavaBeans specification. They must be serializable and have the default no-argument constructor, and all properties must be accessible through getter and setter methods.

    Refer to the Sun Microsystems, Inc., Web site at http://java.sun.com for information about the Enterprise JavaBeans specification.

  In addition, the exception must be a checked exception, inherited from java.lang.Exception, and it must be singular (that is, it does not support throwing multiple checked exception types).

  Note also that the business interface of a Java EnterpriseBean is a plain Java interface and must not extend javax.ejb.EJBObject or javax.ejb.EJBLocalObject. The methods of the business interface should not throw java.rmi.Remote.Exception.

The EJB export bindings can interact with Java EE business logic using either the EJB 2.1 programming model or the EJB 3.0 programming model.

The invocation can be local (for EJB 3.0 only) or remote.

- Local invocation is used when the Java EE business logic calls an SCA component that resides on the same server as the export.
- Remote invocation is used when the Java EE business logic does not reside on the same server as the export.

  For example, in the following figure, an EJB uses RMI/IIOP to call an SCA component on a different server.

Figure 296. Remote call from a client to an SCA component by way of an EJB export

When it configures the EJB binding, Integration Designer uses the JNDI name to determine the EJB programming model level and the type of invocation (local or remote).

EJB export bindings contain the following major components:

- JAX-WS data handler
- EJB export function selector

If your user scenario is not based on the JAX-WS mapping, you might need a custom data handler and function selector to perform the tasks otherwise completed by the components that are part of the EJB export bindings. This includes the mapping normally completed by the custom mapping algorithm.

EJB binding properties:

EJB import bindings use their configured JNDI names to determine the EJB programming model level and type of invocation (local or remote). EJB import and export bindings use the JAX-WS data handler for data transformation. The EJB import binding uses an EJB import function selector and an EJB fault selector, and the EJB export binding uses an EJB export function selector.

JNDI names and EJB import bindings:

When it configures the EJB binding on an import, Integration Designer uses the JNDI name to determine the EJB programming model level and type of invocation (local or remote).

If no JNDI name is specified, the default EJB interface binding is used. The default names that are created depend on whether you are invoking EJB 2.1 JavaBeans or EJB 3.0 JavaBeans.

Note: Refer to the WebSphere Application Server information center for more detailed information about naming conventions: EJB 3.0 application bindings overview.

- EJB 2.1 JavaBeans

  The default JNDI name preselected by Integration Designer is the default EJB 2.1 binding, which takes the form `ejb/` plus the home interface, separated by slashes.

  For example, for the home interface of EJB 2.1 JavaBeans for com.mycompany.myremotebusinesshome, the default binding is:

  ```
  ejb/com/mycompany/myremotebusinesshome
  ```

  For EJB 2.1, only remote EJB invocation is supported.

- EJB 3.0 JavaBeans

  The default JNDI name preselected by Integration Designer for the local JNDI is the fully qualified class name of the local interface preceded by `ejblocal:`. For example, for the fully qualified interface of the local interface com.mycompany.mylocalbusiness, the preselected EJB 3.0 JNDI is:

  ```
  ejblocal:com.mycompany.mylocalbusiness
  ```

  For the remote interface com.mycompany.myremotebusiness, the preselected EJB 3.0 JNDI is the fully qualified interface:

  ```
  com.mycompany.myremotebusiness
  ```

  The EJB 3.0 default application bindings are described at the following location: EJB 3.0 application bindings overview.

  Integration Designer will use the "short" name as the default JNDI location for EJBs using the version 3.0 programming model.

  Note: If the deployed JNDI reference of the target EJB is different from the default JNDI binding location because a custom mapping was used or configured, the target JNDI name must be properly specified. You can specify

the name in Integration Designer before deployment, or, for the import binding, you can change the name in the administrative console (after deployment) to match the JNDI name of the target EJB.

For more information on creating EJB bindings, see the section devoted to Working with EJB bindings in the Integration Designer information center.

JAX-WS data handler:

The Enterprise JavaBeans (EJB) import binding uses the JAX-WS data handler to turn request business objects into Java object parameters and to turn the Java object return value into the response business object. The EJB export binding uses the JAX-WS data handler to turn request EJBs into request business objects and to turn the response business object into a return value.

This data handler maps data from the SCA-specified WSDL interface to the target EJB Java interface (and vice versa) using the Java API for XML Web Services (JAX-WS) specification and the Java Architecture for XML Binding (JAXB) specification.

Note: Current support is restricted to the JAX-WS 2.1.1 and JAXB 2.1.3 specifications.

The data handler specified at the EJB binding level is used to perform request, response, fault, and runtime exception processing.

Note: For faults, a specific data handler can be specified for each fault by specifying the faultBindingType configuration property. This overrides the value specified at the EJB binding level.

The JAX-WS data handler is used by default when the EJB binding has a WSDL interface. This data handler cannot be used to transform a SOAP message representing a JAX-WS invocation to a data object.

The EJB import binding uses a data handler to transform a data object into a Java Object array (Object[]). During outbound communications, the following processing takes place:

1. The EJB binding sets the expected type, expected element, and targeted method name in the BindingContext to match those specified in the WSDL.
2. The EJB binding invokes the transform method for the data object requiring data transformation.
3. The data handler returns an Object[] representing the parameters of the method (in the order of their definition within the method).
4. The EJB binding uses the Object[] to invoke the method on the target EJB interface.

The binding also prepares an Object[] to process the response from the EJB invocation.

- The first element in the Object[] is the return value from the Java method invocation.
- The subsequent values represent the input parameters for the method.

This is required to support the In/Out and Out types of parameters.

For parameters of type Out, the values must be returned in the response data object.

The data handler processes and transforms values found in the Object[] and then returns a response to the data object.

The data handler supports xs:AnyType, xs:AnySimpleType, and xs:Any along with other XSD data types. To enable support for xs:Any, use the `@XmlAnyElement (lax=true)` for the JavaBeans property in the Java code, as shown in the following example:

```
public class TestType {
 private Object[] object;

 @XmlAnyElement (lax=true)
 public Object[] getObject() {
  return object;
 )

 public void setObject (Object[] object) {
  this.object=object;
 )
)
```

This makes the property object in TestType an xs:any field. The Java class value used in the xs:any field should have the `@XmlAnyElement` annotation. For example, if Address is the Java class being used to populate the object array, the Address class should have the annotation `@XmlRootElement`.

Note: To customize the mapping from the XSD type to Java types defined by the JAX-WS specification, change the JAXB annotations to fit your business need. The JAX-WS data handler supports xs:any, xs:anyType, and xs:anySimpleType.

The following restrictions are applicable for the JAX-WS data handler:

- The data handler does not include support for the header attribute `@WebParam` annotation.
- The namespace for business object schema files (XSD files) does not include default mapping from the Java package name. The annotation `@XMLSchema` in package-info.java also does not work. The only way to create an XSD with a namespace is to use the `@XmlType` and `@XmlRootElement` annotations. `@XmlRootElement` defines the target namespace for the global element in JavaBeans types.
- The EJB import wizard does not create XSD files for unrelated classes. Version 2.0 does not support the `@XmlSeeAlso` annotation, so if the child class is not referenced directly from the parent class, an XSD is not created. The solution to this problem is to run SchemaGen for such child classes.

  SchemaGen is a command line utility (located in the *WPS_Install_Home*/bin directory) provided to create XSD files for a given bean. These XSDs must be manually copied to the module for the solution to work.

EJB fault selector:

The EJB fault selector determines if an EJB invocation has resulted in a fault, a runtime exception, or a successful response.

If a fault is detected, the EJB fault selector returns the native fault name to the binding runtime so the JAX-WS data handler can convert the exception object into a fault business object.

On a successful (non-fault) response, the EJB import binding assembles a Java object array (Object[]) to return the values.

- The first element in the Object[] is the return value from the Java method invocation.
- The subsequent values represent the input parameters for the method.

This is required to support the In/Out and Out types of parameters.

For exception scenarios, the binding assembles an Object[] and the first element represents the exception thrown by the method.

The fault selector can return any of the following values:

Table 81. Return values

Type	Return value	Description
Fault	`ResponseType.FAULT`	Returned when the passed Object[] contains an exception object.
Runtime exception	`ResponseType.RUNTIME`	Returned if the exception object does not match any of the declared exception types on the method.
Normal response	`ResponseType.RESPONSE`	Returned in all other cases.

If the fault selector returns a value of `ResponseType.FAULT`, the native fault name is returned. This native fault name is used by the binding to determine the corresponding WSDL fault name from the model and to invoke the correct fault data handler.

EJB function selector:

The EJB bindings use an import function selector (for outbound processing) or an export function selector (for inbound processing) to determine the EJB method to call.

Import function selector

For outbound processing, the import function selector derives the EJB method type based on the name of the operation invoked by the SCA component that is wired to the EJB import. The function selector looks for the @WebMethod annotation on the Integration Designer-generated JAX-WS annotated Java class to determine the associated target operation name.

- If the @WebMethod annotation is present, the function selector uses the @WebMethod annotation to determine the correct Java method mapping for the WSDL method.
- If the @WebMethod annotation is missing, the function selector assumes that the Java method name is the same as the invoked operation name.

Note: This function selector is valid only for a WSDL-typed interface on an EJB import, not for a Java-typed interface on an EJB import.

The function selector returns a java.lang.reflect.Method object that represents the method of the EJB interface.

The function selector uses a Java Object array (Object[]) to contain the response from the target method. The first element in the Object[] is a Java method with the name of the WSDL, and the second element in the Object[] is the input business object.

Export function selector

For inbound processing, the export function selector derives the target method to be invoked from the Java method.

The export function selector maps the Java operation name invoked by the EJB client into the name of the operation in the interface of the target component. The method name is returned as a String and is resolved by the SCA runtime depending on the interface type of the target component.

EIS bindings

Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved using EIS exports and EIS imports that support JCA 1.5 resource adapters and Websphere Adapters.

Your SCA components might require that data be transferred to or from an external EIS. When you create an SCA module requiring such connectivity, you will include (in addition to your SCA component) an import or export with an EIS binding for communication with a specific external EIS.

Resource adapters in IBM Integration Designer are used within the context of an import or an export. You develop an import or an export with the external service wizard and, in developing it, include the resource adapter. An EIS import, which lets your application invoke a service on an EIS system, or an EIS export, which lets an application on an EIS system invoke a service developed in IBM Integration Designer, are created with a resource adapter. For example, you would create an import with the JD Edwards adapter to invoke a service on the JD Edwards system.

When you use the external service wizard, the EIS binding information is created for you. You can also use another tool, the assembly editor, to add or modify binding information. See the Accessing external services with adapters for more information.

After the SCA module containing the EIS binding is deployed to the server, you can use the administrative console to view information about the binding or to configure the binding.

EIS bindings overview:

The EIS (enterprise information system) binding, when used with a JCA resource adapter, lets you access services on an enterprise information system or make your services available to the EIS.

The following example shows how an SCA module named ContactSyncModule synchronizes contact information between a Siebel system and an SAP system.

1. The SCA component named ContactSync listens (by way of an EIS application export named Siebel Contact) for changes to Siebel contacts.
2. The ContactSync SCA component itself makes use of an SAP application (through an EIS application import) in order to update the SAP contact information accordingly.

Because the data structures used for storing contacts are different in Siebel and SAP systems, the ContactSync SCA component must provide mapping.

Figure 297. Flow from a Siebel system to an SAP system

The Siebel Contact export and the SAP Contact import have the appropriate resource adapters configured.

Key features of EIS bindings:

An EIS import is a Service Component Architecture (SCA) import that allows components in the SCA module to use EIS applications defined outside the SCA module. An EIS import is used to transfer data from the SCA component to an external EIS; an EIS export is used to transfer data from an external EIS into the SCA module.

Imports

The role of the EIS import is to bridge the gap between SCA components and external EIS systems. External applications can be treated as an EIS import. In this case, the EIS import sends data to the external EIS and optionally receives data in response.

The EIS import provides SCA components with a uniform view of the applications external to the module. This allows components to communicate with an external EIS, such as SAP, Siebel, or PeopleSoft, using a consistent SCA model.

On the client side of the import, there is an interface, exposed by the EIS import application, with one or more methods, each taking data objects as arguments and return values. On the implementation side, there is a Common Client Interface (CCI) implemented by a resource adapter.

The runtime implementation of an EIS import connects the client-side interface and the CCI. The import maps the invocation of the method on the interface to the invocation on the CCI.

Bindings are created at three levels: the interface binding, which then uses the contained method bindings, which in turn use data bindings.

The interface binding relates the interface of the import to the connection to the EIS system providing the application. This reflects the fact that the set of applications, represented by the interface, is provided by the specific instance of the EIS and the connection provides access to this instance. The binding element contains properties with enough information to create the connection (these properties are part of the javax.resource.spi.ManagedConnectionFactory instance).

The method binding associates the method with the specific interaction with the EIS system. For JCA, the interaction is characterized by the set of properties of the javax.resource.cci.InteractionSpec interface implementation. The interaction element of the method binding contains these properties, along with the name of the class, thus providing enough information to perform the interaction. The method binding uses data bindings describing the mapping of the argument and result of the interface method to EIS representation.

The runtime scenario for an EIS import is as follows:

1. The method on the import interface is invoked using the SCA programming model.
2. The request, reaching the EIS import, contains the name of the method and its arguments.
3. The import first creates an interface binding implementation; then, using data from the import binding, it creates a ConnectionFactory and associates the two. That is, the import calls setConnectionFactory on the interface binding.
4. The method binding implementation matching the invoked method is created.
5. The javax.resource.cci.InteractionSpec instance is created and populated; then, data bindings are used to bind the method arguments to a format understood by the resource adapter.
6. The CCI interface is used to perform the interaction.
7. When the call returns, the data binding is used to create the result of the invocation, and the result is returned to the caller.

Exports

The role of the EIS export is to bridge the gap between an SCA component and an external EIS. External applications can be treated as an EIS export. In this case, the external application sends its data in the form of periodic notifications. An EIS export can be thought of as a subscription application listening to an external request from an EIS. The SCA component that uses the EIS export views it as a local application.

The EIS export provides SCA components with a uniform view of the applications external to the module. This allows components to communicate with an EIS, such as SAP, Siebel, or PeopleSoft, using a consistent SCA model.

The export features a listener implementation receiving requests from the EIS. The listener implements a resource adapter-specific listener interface. The export also contains a component implementing interface, exposed to the EIS through the export.

The runtime implementation of an EIS export connects the listener with the component implementing interface. The export maps the EIS request to the invocation of the appropriate operation on the component. Bindings are created at three levels: a listener binding, which then uses a contained native method binding, which in turn uses a data binding .

The listener binding relates the listener receiving requests with the component exposed through the export. The export definition contains the name of the component; the runtime locates it and forwards requests to it.

The native method binding associates the native method or the event type received by the listener to the operation implemented by the component exposed by way of the export. There is no relationship between the method invoked on the listener and the event type; all the events arrive through one or more methods of the listener. The native method binding uses the function selector defined in the export to extract the native method name from the inbound data and data bindings to bind the data format of the EIS to a format understood by the component.

The runtime scenario for an EIS export is as follows:

1. The EIS request triggers invocation of the method on the listener implementation.
2. The listener locates and invokes the export, passing to it all the invocation arguments.
3. The export creates the listener binding implementation.
4. The export instantiates the function selector and sets it on the listener binding.
5. The export initializes native method bindings and adds them to the listener binding. For each native method binding, the data bindings are also initialized.
6. The export invokes the listener binding.
7. The listener binding locates exported components and uses the function selector to retrieve the native method name.
8. This name is used to locate the native method binding, which then invokes the target component.

The adapter interaction style allows for the EIS export binding to invoke the target component either asynchronously (the default) or synchronously.

Resource adapters

You develop an import or an export with the external service wizard and, in developing it, include a resource adapter. The adapters that come with IBM Integration Designer used to access CICS, IMS, JD Edwards, PeopleSoft, SAP and Siebel systems are intended for development and test purposes only. This means you use them to develop and test your applications.

Once you deploy your application, you will need licensed runtime adapters to run your application. However, when you build your service you can embed the adapter with your service. Your adapter licensing might allow you to use the embedded adapter as the licensed runtime adapter. These adapters are compliant with the Java EE Connector Architecture (JCA 1.5). JCA, an open standard, is the Java EE standard for EIS connectivity. JCA provides a managed framework; that is, Quality of Service (QoS) is provided by the application server, which offers life-cycle management and security to transactions. They are also compliant with the Enterprise Metadata Discovery specification with the exception of IBM CICS ECI Resource Adapter and IBM IMS Connector for Java.

The WebSphere Business Integration Adapters, an older set of adapters, are also supported by the wizard.

Java EE resources

The EIS module, an SCA module that follows the EIS module pattern, can be deployed to the Java EE platform.

The deployment of the EIS module to the Java EE platform results in an application that is ready to execute, packaged as an EAR file and deployed to the server. All Java EE artifacts and resources are created; the application is configured and ready to be run.

JCA Interaction Spec and Connection Spec dynamic properties:

The EIS binding can accept input for the InteractionSpec and ConnectionSpec specified by using a well-defined child data object that accompanies the payload. This allows for dynamic request-response interactions with a resource adapter through the InteractionSpec and component authentication through the ConnectionSpec.

The javax.cci.InteractionSpec carries information on how the interaction request with the resource adapter should be handled. It can also carry information on how the interaction was achieved after the request. These two-way communications through the interactions are sometimes referred to as *conversations*.

The EIS binding expects the payload that will be the argument to the resource adapter to contain a child data object called `properties`. This property data object will contain name/value pairs, with the name of the Interaction Spec properties in a specific format. The formatting rules are:

- Names must begin with the prefix `IS`, followed by the property name. For example, an interaction spec with a JavaBeans property called `InteractionId` would specify the property name as `ISInteractionId`.
- The name/value pair represents the name and the value of the simple type of the Interaction Spec property.

In this example, an interface specifies that the input of an operation is an `Account` data object. This interface invokes an EIS import binding application with the intention to send and receive a dynamic InteractionSpec property called `workingSet` with the value `xyz`.

The business graph or business objects in the server contain an underlying `properties` business object that permits the sending of protocol-specific data with the payload. This `properties` business object is built-in and does not need to be specified in the XML schema when constructing a business object. It only needs to be created and used. If you have your own data types defined based on an XML schema, you need to specify a `properties` element that contains your expected name/value pairs.

```
BOFactory dataFactory = (BOFactory) \
serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
 //Wrapper for doc-lit wrapped style interfaces,
 //skip to payload for non doc-lit
 DataObject docLitWrapper = dataFactory.createByElement /
 ("http://mytest/eis/Account", "AccountWrapper");
```

Create the payload.

```
DataObject account = docLitWrapper.createDataObject(0);
  DataObject accountInfo = account.createDataObject("AccountInfo");
  //Perform your setting up of payload
```

```
//Construct properties data for dynamic interaction

DataObject properties = account.createDataObject("properties");
```

For name workingSet, set the value expected (xyz).

```
properties.setString("ISworkingSet", "xyz");
```

```
//Invoke the service with argument

Service accountImport = (Service) \
serviceManager.locateService("AccountOutbound");
DataObject result = accountImport.invoke("createAccount", docLitWrapper);

//Get returned property
DataObject retProperties = result.getDataObject("properties");

String workingset = retProperties.getString("ISworkingSet");
```

You can use ConnectionSpec properties for dynamic component authentication. The same rules apply as above, except that the property name prefix needs to be `CS` (instead of `IS`). ConnectionSpec properties are not two-way. The same `properties` data object can contain both IS and CS properties.

To use ConnectionSpec properties, set the `resAuth` specified on the import binding to `Application`. Also, make sure the resource adapter supports component authorization. See chapter 8 of the J2EE Connector Architecture Specification for more details.

External clients with EIS bindings:

The server can send messages to, or receive messages from, external clients using EIS bindings.

An external client, for example a Web portal or an EIS, needs to send a message to an SCA module in the server or needs to be invoked by a component from within the server.

The client invokes the EIS import as with any other application, using either the Dynamic Invocation Interface (DII) or Java interface.

1. The external client creates an instance of the ServiceManager and looks up the EIS import using its reference name. The result of the lookup is a service interface implementation.
2. The client creates an input argument, a generic data object, created dynamically using the data object schema. This step is done using the Service Data Object DataFactory interface implementation.
3. The external client invokes the EIS and obtains the required results.

Alternatively, the client can invoke the EIS import using the Java interface.

1. The client creates an instance of the ServiceManager and looks up the EIS import using its reference name. The result of the lookup is a Java interface of the EIS import.
2. The client creates an input argument and a typed data object.
3. The client invokes EIS and obtains the required results.

The EIS export interface defines the interface of the exported SCA component that is available to the external EIS applications. This interface can be thought of as the interface that an external application (such as SAP or PeopleSoft) will invoke through the implementation of the EIS export application runtime.

The export uses EISExportBinding to bind exported services to the external EIS application. It allows you to subscribe an application contained in your SCA module to listen for EIS service requests. The EIS export binding specifies the mapping between the definition of inbound events as it is understood by the resource adapter (using Java EE Connector Architecture interfaces) and the invocation of SCA operations.

The EISExportBinding requires external EIS services to be based on Java EE Connector Architecture 1.5 inbound contracts. The EISExportBinding requires that a data handler or data binding be specified either at the binding level or the method level.

JMS bindings

A Java Message Service (JMS) provider enables messaging based on the Java Messaging Service API and programming model. It provides Java EE connection factories to create connections for JMS destinations and to send and receive messages.

Three JMS bindings are provided:

- Service integration bus (SIB) provider binding compliant with JMS JCA 1.5 (*JMS binding*)
- Generic, non-JCA JMS binding, compliant with JMS 1.1 (*Generic JMS binding*)
- WebSphere MQ JMS binding, providing JMS provider support for WebSphere MQ and allowing Java EE application interoperability (*WebSphere MQ JMS binding*)

The JMS export and import bindings allow a Service Component Architecture (SCA) module to make calls to, and receive messages from, external JMS systems.

Also supported are WebSphere MQ bindings (*WebSphere MQ binding*) which allow native MQ users to handle arbitrary incoming and outgoing message formats (WebSphere MQ required).

The JMS import and export bindings provide integration with JMS applications using the JCA 1.5-based SIB JMS provider that is part of WebSphere Application Server. Other JCA 1.5–based JMS resource adapters are not supported

JMS bindings overview:

JMS bindings provide connectivity between the Service Component Architecture (SCA) environment and JMS systems.

JMS bindings

The major components of both JMS import and JMS export bindings are:

- Resource adapter: enables managed, bidirectional connectivity between an SCA module and external JMS systems
- Connections: encapsulate a virtual connection between a client and a provider application

- Destinations: used by a client to specify the target of messages it produces or the source of messages it consumes
- Authentication data: used to secure access to the binding

JMS import bindings

JMS import bindings allow you to import an external JMS application to be used inside your SCA module. JMS import bindings allow components within your SCA module to communicate with services provided by external JMS applications.

Connections to the associated JMS provider of JMS destinations are created by using a JMS connection factory. Use connection factory administrative objects to manage JMS connection factories for the default messaging provider.

Interaction with external JMS systems includes the use of destinations for sending requests and receiving replies.

Two types of usage scenarios for the JMS import binding are supported, depending on the type of operation being invoked:

- One-way: The JMS import puts a message on the send destination configured in the import binding. Nothing is set in the replyTo field of the JMS header.
- Two-way (request-response): The JMS import puts a message on the send destination and then persists the reply it receives from the SCA component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default), or from the request message correlation ID. The import binding can also be configured to use a temporary dynamic response destination to correlate responses with requests. A temporary destination is created for each request and the import uses this destination to receive the response.

  The receive destination is set in the replyTo header property of the outbound message. A message listener is deployed to listen on the receive destination, and when a reply is received, the message listener passes the reply back to the component.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified. Static properties can be set from the JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that JMS is an asynchronous binding. If a calling component invokes a JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 44 on page 124 illustrates how the import is linked to the external service.

Figure 298. JMS import binding resources

JMS export bindings

JMS export bindings provide the means for SCA modules to provide services to external JMS applications.

The connection that is part of a JMS export is a configurable activation specification.

A JMS export has send and receive destinations.

- The receive destination is where the incoming message for the target component should be placed.
- The send destination is where the reply will be sent, unless the incoming message has overridden this using the replyTo header property.

A message listener is deployed to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply. The destination specified in the replyTo field of the incoming message overrides the destination specified in the send.

Figure 45 on page 125 illustrates how the external requestor is linked to the export.

Figure 299. JMS export binding resources

JMS integration and resource adapters:

The Java Message Service (JMS) provides integration through an available JMS JCA 1.5-based resource adapter. Complete support for JMS integration is provided for the Service Integration Bus (SIB) JMS resource adapter.

Use a JMS provider for JCA 1.5 resource adapter when you want to integrate with an external JCA 1.5-compliant JMS system. External services compliant with JCA 1.5 can receive messages and send messages to integrate with your service component architecture (SCA) components using the SIB JMS resource adapter.

The use of other provider-specific JCA 1.5 resource adapters is not supported.

JMS modules cannot be deployed to a Java SE environment. Such modules are only deployable to a Java EE environment.

Key features of JMS bindings:

Key features of JMS import and export bindings include headers and created Java EE resources.

Special headers

Special header properties are used in JMS imports and exports to tell the target how to handle the message.

For example, TargetFunctionName maps from the native method to the operation method.

Java EE resources

A number of Java EE resources are created when JMS imports and exports are deployed to a Java EE environment.

`ConnectionFactory`
Used by clients to create a connection to the JMS provider.

`ActivationSpec`
Imports use this for receiving the response to a request; exports use it when configuring the message endpoints that represent message listeners in their interactions with the messaging system.

`Destinations`

- Send destination: on an import, this is where the request or outgoing message is sent; on an export, this is the destination where the response message will be sent, if not superseded by the JMSReplyTo header field in the incoming message.
- Receive destination: where the incoming message should be placed; with imports, this is a response; with exports, this is a request.
- Callback destination: SCA JMS system destination used to store correlation information. Do not read or write to this destination.

The installation task creates the ConnectionFactory and three destinations. It also creates the ActivationSpec to enable the runtime message listener to listen for replies on the receive destination. The properties of these resources are specified in the import or export file.

JMS headers:

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

JMS system header

The JMS system header is represented in the SMO by the JMSHeader element, which contains all the fields typically found in a JMS header. Although these can be modified in the mediation (or ContextService), some JMS system header fields set in the SMO will not be propagated in the outbound JMS message as they are overridden by system or static values.

The key fields in the JMS system header that can be updated in a mediation (or ContextService) are:

- **JMSType** and **JMSCorrelationID** – values of the specific predefined message header properties
- **JMSDeliveryMode** – values for delivery mode (persistent or nonpersistent; default is persistent)

- **JMSPriority** – priority value (0 to 9; default is JMS_Default_Priority)

JMS properties

JMS properties are represented in the SMO as entries in the Properties list. The properties can be added, updated, or deleted in a mediation or by using the ContextService API.

Properties can also be set statically in the JMS binding. Properties that are set statically override settings (with the same name) that are set dynamically.

User properties propagated from other bindings (for example, an HTTP binding) will be output in the JMS binding as JMS properties.

Header propagation settings

The propagation of the JMS system header and properties either from the inbound JMS message to downstream components or from upstream components to the outbound JMS message can be controlled by the Propagate Protocol Header flag on the binding.

When Propagate Protocol Header is set, header information is allowed to flow to the message or to the target component, as described in the following list:

- JMS export request

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.
- JMS export response

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS export binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS export binding.
- JMS import request

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS import binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS import binding.
- JMS import response

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.

JMS temporary dynamic response destination correlation scheme:

The temporary dynamic response destination correlation scheme causes a unique dynamic queue or topic to be created for each request sent.

The static response destination specified in the import is used to derive the nature of the temporary dynamic destination queue or topic. This is set in the **ReplyTo** field of the request, and the JMS import listens for responses on that destination. When the response is received it is requeued to the static response destination for asynchronous processing. The **CorrelationID** field of the response is not used, and does not need to be set.

Transactional issues

When a temporary dynamic destination is being used, the response must be consumed in the same thread as the sent response. The request must be sent outside the global transaction, and must be committed before it is received by the backend service, and a response returned.

Persistence

Temporary dynamic queues are short-lived entities and do not guarantee the same level of persistence associated with a static queue or topic. A temporary dynamic queue or topic will not survive a server restart and neither will messages. After the message has been requeued to the static response destination it retains the persistence defined in the message.

Timeout

The import waits to receive the response on the temporary dynamic response destination for a fixed amount of time. This time interval will be taken from the SCA Response Expiration time qualifier, if it is set, otherwise the time defaults to 60 seconds. If the wait time is exceeded the import throws a `ServiceTimeoutRuntimeException`.

External clients:

The server can send messages to, or receive messages from, external clients using JMS bindings.

An external client (such as a Web portal or an enterprise information system) can send a message to an SCA module in the server, or it can be invoked by a component from within the server.

The JMS export components deploy message listeners to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications with the export binding.

JMS imports bind to, and can deliver messages to, external clients. This message might or might not demand a response from the external client.

Working with external clients:

An external client (that is, outside the server) might need to interact with an application installed in the server.

About this task

Consider a very simple scenario in which an external client wants to interact with an application on the server. The figure depicts a typical simple scenario.

Figure 300. Simple use-case scenario: external client interacts with server application

The SCA application includes an export with a JMS binding; this makes the application available to external clients.

When you have an external client in a Java virtual machine (JVM) separate from your server, there are several steps you must take in order to make a connection and interact with a JMS export. The client obtains an InitialContext with the correct values and then looks up the resources through JNDI. The client then uses the JMS 1.1 specification client to access the destinations and the send and receive messages on the destinations.

The default JNDI names of the resources created automatically by the runtime are listed in the configuration topic of this section. However, if you have pre-created resources, use those JNDI names.

Procedure

1. Configure JMS destinations and the connection factory to send the message.
2. Make sure that the JNDI context, the port for the SIB resource adapter, and the messaging bootstrapping port are correct.

   The server uses some default ports, but if there are more servers installed on that system, alternate ports are created at installation time to avoid conflicts with other server instances. You can use the administrative console to determine which ports your server is employing . Go to **Servers** > **Application Servers** > ***your_server_ name*** > **Configuration** and click **Ports** under **Communication**. You can then edit the port being used.
3. The client obtains an initial context with the correct values and then looks up the resources through JNDI.
4. Using JMS 1.1 specifications, the client accesses the destinations and the send and receive messages on the destinations.

Troubleshooting JMS bindings:

You can diagnose and fix problems with JMS bindings.

Implementation exceptions

In response to various error conditions, the JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

  For example, an import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

Primary failure conditions

The primary failure conditions of JMS bindings are determined by transactional semantics, by JMS provider configuration, or by reference to existing behavior in other components. The primary failure conditions include:

- Failure to connect to the JMS provider or destination.

  A failure to connect to the JMS provider to receive messages will result in the message listener failing to start. This condition will be logged in the WebSphere Application Server log. Persistent messages will remain on the destination until they are successfully retrieved (or expired).

  A failure to connect to the JMS provider to send outbound messages will cause rollback of the transaction controlling the send.
- Failure to parse an inbound message or to construct an outbound message.

  A failure in the data binding or data handler causes rollback of the transaction controlling the work.
- Failure to send the outbound message.

  A failure to send a message causes rollback of the relevant transaction.
- Multiple or unexpected late response messages.

  The import expects only one response message for each request message. Also the valid time period in which a response can be received is determined by the SCA Response Expiration qualifier on the request. When a response arrives or the expiration time is exceeded, the correlation record is deleted. If response messages arrive unexpectedly or arrive late, a Service Runtime Exception is thrown.
- Service timeout runtime exception caused by late response when using the temporary dynamic response destination correlation scheme.

  The JMS import will timeout after a period of time determined by the SCA response expiration qualifier, or if this is not set it will default to 60 seconds.

JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the underlying SIB destination of the JMS destination has a maximum failed deliveries value greater than 1. Setting this value to 2 or more enables interaction with the failed event manager during SCA invocations for the JMS bindings.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a `DataBindingException`.

When any runtime exception, including a `DataBindingException`, is thrown:

- If the mediation flow is configured to be transactional, the JMS message , by default, is stored in the Failed Event Manager for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the Failed Event Manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a `DataHandlerException` is reported to you as a `DataBindingException`.

Generic JMS bindings

The Generic JMS binding provides connectivity to third-party JMS 1.1 compliant providers. The operation of the Generic JMS bindings is similar to that of JMS bindings.

The service provided through a JMS binding allows a Service Component Architecture (SCA) module to make calls or receive messages from external systems. The system can be an external JMS system.

The Generic JMS binding provides integration with non-JCA 1.5-compliant JMS providers that support JMS 1.1 and implement the optional JMS Application Server Facility. The Generic JMS binding supports those JMS providers (including Oracle AQ, TIBCO, SonicMQ, WebMethods, BEA WebLogic, and WebSphere MQ) that do not support JCA 1.5 but do support the Application Server Facility of the JMS 1.1 specification. The WebSphere embedded JMS provider (SIBJMS), which is a JCA 1.5 JMS provider, is not supported by this binding; when using that provider, use the "JMS bindings" on page 122.

Use this Generic binding when integrating with a non-JCA 1.5-compliant JMS-based system within an SCA environment. The target external applications can then receive messages and send messages to integrate with an SCA component.

Generic JMS bindings overview:

Generic JMS bindings are non-JCA JMS bindings that provide connectivity between the Service Component Architecture (SCA) environment and JMS systems that are compliant with JMS 1.1 and that implement the optional JMS Application Server Facility.

Generic JMS bindings

The major aspects of Generic JMS import and export bindings are:

- Listener port: enables non-JCA-based JMS providers to receive messages and dispatch them to a Message Driven Bean (MDB)
- Connections: encapsulate a virtual connection between a client and a provider application
- Destinations: used by a client to specify the target of messages it produces or the source of messages it consumes
- Authentication data: used to secure access to the binding

Generic JMS import bindings

Generic JMS import bindings allow components within your SCA module to communicate with services provided by external non-JCA 1.5-compliant JMS providers.

The connection part of a JMS import is a connection factory. A connection factory, the object a client uses to create a connection to a provider, encapsulates a set of connection configuration parameters defined by an administrator. Each connection factory is an instance of the ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory interface.

Interaction with external JMS systems includes the use of destinations for sending requests and receiving replies.

Two types of usage scenarios for the Generic JMS import binding are supported, depending on the type of operation being invoked:

- One-way: The Generic JMS import puts a message on the send destination configured in the import binding Nothing is sent to the replyTo field of the JMS header.
- Two-way (request-response): The Generic JMS import puts a message on the send destination and then persists the reply it receives from the SCA component.

  The receive destination is set in the replyTo header property of the outbound message. A message driven bean (MDB) is deployed to listen on the receive destination, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified. Static properties can be set from the Generic JMS import method binding Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that Generic JMS is an asynchronous binding. If a calling component invokes a Generic JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 47 on page 133 illustrates how the import is linked to the external service.

Figure 301. Generic JMS import binding resources

Generic JMS export bindings

Generic JMS export bindings provide the means for SCA modules to provide services to external JMS applications.

The connection part of a JMS export is composed of a ConnectionFactory and a ListenerPort.

A Generic JMS export has send and receive destinations.

- The receive destination is where the incoming message for the target component should be placed.
- The send destination is where the reply will be sent, unless the incoming message has overridden this using the replyTo header property.

An MDB is deployed to listen to requests incoming to the receive destination specified in the export binding.

- The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply.
- The destination specified in the replyTo field of the incoming message overrides the destination specified in the send field.
- For request/response scenarios, the import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response to copy the request message ID to the correlation ID field of the response message (default), or the response can copy the request correlation ID to the correlation ID field of the response message.

Figure 48 on page 134 illustrates how the external requestor is linked to the export.

Figure 302. Generic JMS export binding resources

Key features of Generic JMS bindings:

The features of the Generic JMS import and export binding are consistent with those of the WebSphere embedded JMS and MQ JMS import bindings. Key features include header definitions and access to existing Java EE resources. However, because of its generic nature, there are no JMS provider-specific connectivity options, and this binding has limited capability to generate resources at deployment and installation.

Generic imports

Like the MQ JMS import application, the Generic JMS implementation is asynchronous and supports three invocations: one-way, two-way (also known as request-response), and callback.

When the JMS import is deployed, a message driven bean (MDB) provided by the runtime environment is deployed. The MDB listens for replies to the request message. The MDB is associated with (listens on) the destination sent with the request in the replyTo header field of the JMS message.

Generic exports

Generic JMS export bindings differ from EIS export bindings in their handling of the return of the result. A Generic JMS export explicitly sends the response to the replyTo destination specified on the incoming message. If none is specified, the send destination is used.

When the Generic JMS export is deployed, a message driven bean (a different MDB than the one used for Generic JMS imports) is deployed. It listens for the incoming requests on the receive destination and then dispatches the requests to be processed by the SCA runtime.

Special headers

Special header properties are used in Generic JMS imports and exports to tell the target binding how to handle the message.

For example, the TargetFunctionName property is used by the default function selector to identify the name of the operation in the export interface that is being invoked.

Note: The import binding can be configured to set the TargetFunctionName header to the operation name for each operation.

Java EE resources

A number of Java EE resources are created when a JMS binding is deployed to a Java EE environment.

- Listener port for listening on the receive (response) destination (two-way only) for imports and on the receive (request) destination for exports
- Generic JMS connection factory for the outboundConnection (import) and inboundConnection (export)
- Generic JMS destination for the send (import) and receive (export; two-way only) destinations
- Generic JMS connection factory for the responseConnection (two-way only and optional; otherwise, outboundConnection is used for imports, and inboundConnection is used for exports)
- Generic JMS destination for the receive (import) and send (export) destination (two-way only)
- Default messaging provider callback JMS destination used to access the SIB callback queue destination (two-way only)
- Default messaging provider callback JMS connection factory used to access the callback JMS destination (two-way only)
- SIB callback queue destination used to store information about the request message for use during response processing (two-way only)

The installation task creates the ConnectionFactory, the three destinations, and the ActivationSpec from the information in the import and export files.

Generic JMS headers:

Generic JMS headers are Service Data Objects (SDO) that contain all the properties of the Generic JMS message properties. These properties can be from the inbound message or they can be the properties that will be applied to the outbound message.

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

The following properties are set statically on the methodBinding:

- JMSType
- JMSCorrelationID
- JMSDeliveryMode
- JMSPriority

The Generic JMS binding also supports dynamic modification of JMS headers and properties in the same manner as the JMS and MQ JMS bindings.

Some Generic JMS providers place restrictions on which properties can be set by the application and in what combinations. You must consult your third-party product documentation for more information. However, an additional property has been added to the methodBinding, ignoreInvalidOutboundJMSProperties, which allows any exceptions to be propagated.

The Generic JMS headers and message properties are used only when the base service component architecture SCDL binding switch is turned on. When the switch is turned on, context information is propagated. By default, this switch is on. To prevent context information propagation, change the value to `false`.

When context propagation is enabled, header information is allowed to flow to the message or to the target component. To turn on and off context propagation, specify `true` or `false` for the contextPropagationEnabled attribute of the import and export bindings. For example:

```
<esbBinding xsi:type="eis:JMSImportBinding" contextProgagationEnabled="true">
```

The default is `true`.

Troubleshooting Generic JMS bindings:

You can diagnose and fix problems with Generic JMS binding.

Implementation exceptions

In response to various error conditions, the Generic JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

Troubleshooting Generic JMS message expiry

A request message by the JMS provider is subject to expiration.

Request expiry refers to the expiration of a request message by the JMS provider when the JMSExpiration time on the request message is reached. As with other JMS bindings, the Generic JMS binding handles the request expiry by setting expiration on the callback message placed by the import to be the same as for the outgoing request. Notification of the expiration of the callback message will indicate that the request message has expired and the client should be notified by means of a business exception.

If the callback destination is moved to the third-party provider, however, this type of request expiry is not supported.

Response expiry refers to the expiration of a response message by the JMS provider when the JMSExpiration time on the response message is reached.

Response expiry for the generic JMS binding is not supported, because the exact expiry behavior of a third-party JMS provider is not defined. You can, however, check that the response is not expired if and when it is received.

For outbound request messages, the JMSExpiration value will be calculated from the time waited and from the requestExpiration values carried in the asyncHeader, if set.

Troubleshooting Generic JMS connection factory errors

When you define certain types of connection factories in your Generic JMS provider, you might receive an error message when you try to start an application. You can modify the external connection factory to avoid this problem.

When launching an application, you might receive the following error message:

```
MDB Listener Port JMSConnectionFactory type does not match
JMSDestination type
```

This problem can arise when you are defining external connection factories. Specifically, the exception can be thrown when you create a JMS 1.0.2 Topic Connection Factory, instead of a JMS 1.1 (unified) Connection Factory (that is, one that is able to support both point-to-point and publish/subscribe communication).

To resolve this issue, take the following steps:

1. Access the Generic JMS provider that you are using.
2. Replace the JMS 1.0.2 Topic Connection Factory that you defined with a JMS 1.1 (unified) Connection Factory.

When you launch the application with the newly defined JMS 1.1 Connection Factory, you should no longer receive an error message.

Generic JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a generic JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the value of the maximum retries property on the underlying listener port is equal to or greater than 1. Setting this value to 1 or more enables interaction with the failed event manager during SCA invocations for the generic JMS bindings.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a `DataBindingException`.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a `DataHandlerException` is reported to you as a `DataBindingException`.

When any runtime exception, including a `DataBindingException` exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a `DataHandlerException` is reported to you as a `DataBindingException`.

WebSphere MQ JMS bindings

The WebSphere MQ JMS binding provides integration with external applications that use a WebSphere MQ JMS-based provider.

Use the WebSphere MQ JMS export and import bindings to integrate directly with external JMS or MQ JMS systems from your server environment. This eliminates the need to use MQ Link or Client Link features of the Service Integration Bus.

When a component interacts with a WebSphere MQ JMS-based service by way of an import, the WebSphere MQ JMS import binding utilizes a destination to which data will be sent and a destination where the reply can be received. Conversion of the data to and from a JMS message is accomplished through the JMS data handler or data binding edge component.

When an SCA module provides a service to WebSphere MQ JMS clients, the WebSphere MQ JMS export binding utilizes a destination where the request can be received and the response can be sent. The conversion of the data to and from a JMS message is done through the JMS data handler or data binding.

The function selector provides a mapping to the operation on the target component to be invoked.

WebSphere MQ JMS bindings overview:

The WebSphere MQ JMS binding provides integration with external applications that use the WebSphere MQ JMS provider.

WebSphere MQ administrative tasks

The WebSphere MQ system administrator is expected to create the underlying WebSphere MQ Queue Manager, which the WebSphere MQ JMS bindings will use, before running an application containing these bindings.

WebSphere MQ JMS import bindings

The WebSphere MQ JMS import allows components within your SCA module to communicate with services provided by WebSphere MQ JMS-based providers. You must be using a supported version of WebSphere MQ. Detailed hardware and software requirements can be found on the IBM support pages.

Two types of usage scenarios for WebSphere MQ JMS import bindings are supported, depending on the type of operation being invoked:

- One-way: The WebSphere MQ JMS import puts a message on the send destination configured in the import binding. Nothing is sent to the replyTo field of the JMS header.
- Two-way (request-response): The WebSphere MQ JMS import puts a message on the send destination.

  The receive destination is set in the replyTo header field. A message-driven bean (MDB) is deployed to listen on the receive destination, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified.Static properties can be set from the JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that WebSphere MQ JMS is an asynchronous binding. If a calling component invokes a WebSphere MQ JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 49 on page 140 illustrates how the import is linked to the external service.

Figure 303. WebSphere MQ JMS import binding resources

WebSphere MQ JMS export bindings

The WebSphere MQ JMS export binding provides the means for SCA modules to provide services to external JMS applications on the WebSphere MQ-based JMS provider.

An MDB is deployed to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply. The destination specified in the replyTo field of the response message overrides the destination specified in the send field.

Figure 50 on page 141 illustrates how the external requestor is linked to the export.

Figure 304. WebSphere MQ JMS export binding resources

Note: Figure 49 on page 140 and Figure 50 on page 141 illustrate how an application from a previous version of WebSphere ESB is linked to an external service. For applications developed for WebSphere ESB Version 7.0, the Activation specification is used instead of the Listener Port and Connection Factory.

Key features of WebSphere MQ JMS bindings:

Key features of WebSphere MQ JMS bindings include headers, Java EE artifacts, and created Java EE resources.

Headers

A JMS message header contains a number of predefined fields containing values used by both clients and providers to identify and to route messages. You can use binding properties to configure these headers with fixed values, or the headers can be specified dynamically at runtime.

`JMSCorrelationID`
Links to a related message. Typically, this field is set to the message identifier string of the message that is being replied to.

`TargetFunctionName`
This header is used by one of the supplied function selectors to identify the operation being invoked. Setting the TargetFunctionName JMS header property in messages sent to a JMS export allows this function selector to be used. The property can be set directly in JMS client applications or when connecting an import with a JMS binding to such an export. In this case, the JMS import binding should be configured to set the TargetFunctionName header for each operation in the interface to the name of the operation.

Correlation schemes

The WebSphere MQ JMS bindings provide various correlation schemes that are used to determine how to correlate request messages with response messages.

RequestMsgIDToCorrelID
The JMSMessageID is copied to the JMSCorrelationID field. This is the default setting.

RequestCorrelIDToCorrelID
The JMSCorrelationID is copied to the JMSCorrelationID field.

Java EE resources

A number of Java EE resources are created when an MQ JMS import is deployed to a Java EE environment.

Parameters

MQ Connection Factory
Used by clients to create a connection to the MQ JMS provider.

Response Connection Factory
Used by the SCA MQ JMS runtime when the send destination is on a different Queue Manager than the receive destination.

Activation specification
An MQ JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

Destinations

- Send destination:
  - Imports: Where the request or outgoing message is sent.
  - Exports: Where the response message will be sent if it is not superseded by the JMSReplyTo header field of the incoming message.
- Receive destination:
  - Imports: Where the response or incoming message should be placed.
  - Exports: Where the incoming or request message should be placed.

JMS headers:

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

JMS system header

The JMS system header is represented in the SMO by the JMSHeader element, which contains all the fields typically found in a JMS header. Although these can be modified in the mediation (or ContextService), some JMS system header fields set in the SMO will not be propagated in the outbound JMS message as they are overridden by system or static values.

The key fields in the JMS system header that can be updated in a mediation (or ContextService) are:

- **JMSType** and **JMSCorrelationID** – values of the specific predefined message header properties
- **JMSDeliveryMode** – values for delivery mode (persistent or nonpersistent; default is persistent)
- **JMSPriority** – priority value (0 to 9; default is JMS_Default_Priority)

JMS properties

JMS properties are represented in the SMO as entries in the Properties list. The properties can be added, updated, or deleted in a mediation or by using the ContextService API.

Properties can also be set statically in the JMS binding. Properties that are set statically override settings (with the same name) that are set dynamically.

User properties propagated from other bindings (for example, an HTTP binding) will be output in the JMS binding as JMS properties.

Header propagation settings

The propagation of the JMS system header and properties either from the inbound JMS message to downstream components or from upstream components to the outbound JMS message can be controlled by the Propagate Protocol Header flag on the binding.

When Propagate Protocol Header is set, header information is allowed to flow to the message or to the target component, as described in the following list:

- JMS export request

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.
- JMS export response

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS export binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS export binding.
- JMS import request

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS import binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS import binding.
- JMS import response

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.

External clients:

The server can send messages to, or receive messages from, external clients using WebSphere MQ JMS bindings.

An external client (such as a Web portal or an enterprise information system) can send a message to an SCA component in the application by way of an export or it can be invoked by an SCA component in the application by way of an import.

The WebSphere MQ JMS export binding deploys message driven beans (MDBs) to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications via the export binding.

WebSphere MQ JMS imports bind to, and can deliver message to, external clients. This message might or might not demand a response from the external client.

More information on how to interact with external clients using WebSphere MQ can be found at the WebSphere MQ information center.

Troubleshooting WebSphere MQ JMS bindings:

You can diagnose and fix problems with WebSphere MQ JMS bindings.

Implementation exceptions

In response to various error conditions, the MQ JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

  For example, an import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

WebSphere MQ JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a WebSphere MQ JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the value of the maximum retries property on the underlying listener port is equal to or greater than 1. Setting this value to 1 or more enables interaction with the failed event manager during SCA invocations for the MQ JMS bindings.

Misusage scenarios: comparison with WebSphere MQ bindings

The WebSphere MQ JMS binding is designed to interoperate with JMS applications deployed against WebSphere MQ, which exposes messages according to the JMS message model. The WebSphere MQ import and export, however, are principally designed to interoperate with native WebSphere MQ applications and expose the full content of the WebSphere MQ message body to mediations.

The following scenarios should be built using the WebSphere MQ JMS binding, not the WebSphere MQ binding:

- Invoking a JMS message-driven bean (MDB) from an SCA module, where the MDB is deployed against the WebSphere MQ JMS provider. Use a WebSphere MQ JMS import.
- Allowing the SCA module to be called from a Java EE component servlet or EJB by way of JMS. Use a WebSphere MQ JMS export.

- Mediating the contents of a JMS MapMessage, in transit across WebSphere MQ. Use a WebSphere MQ JMS export and import in conjunction with the appropriate data handler or data binding.

There are situations in which the WebSphere MQ binding and WebSphere MQ JMS binding might be expected to interoperate. In particular, when you are bridging between Java EE and non-Java EE WebSphere MQ applications, use a WebSphere MQ export and WebSphere MQ JMS import (or vice versa) in conjunction with appropriate data bindings or mediation modules (or both).

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a `DataBindingException`.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a `DataHandlerException` is reported to you as a `DataBindingException`.

When any runtime exception, including a `DataBindingException` exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a `DataHandlerException` is reported to you as a `DataBindingException`.

WebSphere MQ bindings

The WebSphere MQ binding provides Service Component Architecture (SCA) connectivity with WebSphere MQ applications.

Use the WebSphere MQ export and import bindings to integrate directly with a WebSphere MQ-based system from your server environment. This eliminates the need to use MQ Link or Client Link features of the Service Integration Bus.

When a component interacts with a WebSphere MQ service by way of an import, the WebSphere MQ import binding uses a queue to which data will be sent and a queue where the reply can be received.

When an SCA module provides a service to WebSphere MQ clients, the WebSphere MQ export binding uses a queue where the request can be received and the response can be sent. The function selector provides a mapping to the operation on the target component to be invoked.

Conversion of the payload data to and from an MQ message is done through the MQ body data handler or data binding. Conversion of the header data to and from an MQ message is done through the MQ header data binding.

For information about the WebSphere MQ versions supported, see the system requirements web page: WebSphere ESB system requirements

WebSphere MQ bindings overview:

The WebSphere MQ binding provides integration with native MQ-based applications.

WebSphere MQ administrative tasks

The WebSphere MQ system administrator is expected to create the underlying WebSphere MQ Queue Manager, which the WebSphere MQ bindings will use, before running an application containing these bindings.

WebSphere administrative tasks

You must set the **Native library path** property of the MQ resource adapter in Websphere to the WebSphere MQ version supported by the server, and restart the server. This ensures that the libraries of a supported version of WebSphere MQ are being used. Detailed hardware and software requirements can be found on the IBM support pages

WebSphere MQ import bindings

The WebSphere MQ import binding allows components within your SCA module to communicate with services provided by external WebSphere MQ-based applications. You must be using a supported version of WebSphere MQ. Detailed hardware and software requirements can be found on the IBM support pages.

Interaction with external WebSphere MQ systems includes the use of queues for sending requests and receiving replies.

Two types of usage scenarios for the WebSphere MQ import binding are supported, depending on the type of operation being invoked:

- One-way: The WebSphere MQ import puts a message on the queue configured in the **Send destination queue** field of the import binding. Nothing is sent to the replyTo field of the MQMD header.
- Two-way (request-response): The WebSphere MQ import puts a message on the queue configured in the **Send destination queue** field

  The receive queue is set in the replyTo MQMD header field. A message driven bean (MDB) is deployed to listen on the receive queue, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

It is important to note that WebSphere MQ is an asynchronous binding. If a calling component invokes a WebSphere MQ import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the WebSphere MQ service.

Figure 51 on page 147 illustrates how the import is linked to the external service.

Figure 305. WebSphere MQ import binding resources

WebSphere MQ export bindings

The WebSphere MQ export binding provides the means for SCA modules to provide services to external WebSphere MQ-based applications.

An MDB is deployed to listen to requests incoming to the **Receive destination queue** specified in the export binding. The queue specified in the **Send destination queue** field is used to send the reply to the inbound request if the invoked component provides a reply. The queue specified in the replyTo field of the response message overrides the queue specified in the **Send destination queue** field.

Figure 52 on page 148 illustrates how the external requestor is linked to the export.

Figure 306. WebSphere MQ export binding resources

Note: Figure 51 on page 147 and Figure 52 on page 148 illustrate how an application from a previous version of WebSphere ESB is linked to an external service. For applications developed for WebSphere ESB Version 7.0, the Activation specification is used instead of the Listener Port and Connection Factory.

Key features of a WebSphere MQ binding:

Key features of a WebSphere MQ binding include headers, Java EE artifacts, and created Java EE resources.

Correlation schemes

A WebSphere MQ request/reply application can use one of a number of techniques to correlate response messages with requests, built around the MQMD MessageID and CorrelID fields. In the vast majority of cases, the requestor lets the queue manager select a MessageID and expects the responding application to copy this into the CorrelID of the response. In most cases, the requestor and responding application implicitly know which correlation technique is in use. Occasionally the responding application will honor various flags in the Report field of the request that describe how to handle these fields.

Export bindings for WebSphere MQ messages can be configured with the following options:

Response MsgId options:

New MsgID
Allows the queue manager to select a unique MsgId for the response (default).

Copy from Request MsgID
Copies the MsgId field from the MsgId field in the request.

Copy from SCA message
Sets the MsgId to be that carried in WebSphere MQ headers in the SCA response message, or lets the queue manager define a new Id if the value does not exist.

As Report Options
Inspects the Report field of the MQMD in the request for a hint as to how to handle the MsgId. The MQRO_NEW_MSG_ID and MQRO_PASS_MSG_ID options are supported and behave like New MsgId and Copy from Request MsgID, respectively

Response CorrelId options:

Copy from Request MsgID
Copies the CorrelId field from the MsgId field in the request (default).

Copy from Request CorrelID
Copies the CorrelId field from the CorrelId field in the request.

Copy from SCA message
Sets the CorrelId to be carried in WebSphere MQ headers in the SCA response message or leaves it blank if the value does not exist.

As Report Options
Inspects the Report field of the MQMD in the request for a hint as to how to handle the CorrelId. The MQRO_COPY_MSG_ID_TO_CORREL_ID and MQRO_PASS_CORREL_ID options are supported and behave like Copy from Request MsgID and Copy from Request CorrelID, respectively

Import bindings for WebSphere MQ messages can be configured with the following options:

Request MsgId options:

New MsgID
Allows the queue manager to select a unique MsgId for the request (default)

Copy from SCA message
Sets the MsgId to be carried in WebSphere MQ headers in the SCA request message or lets the queue manager define a new Id if the value does not exist.

Response correlation options:

Response has CorrelID copied from MsgId
Expects the response message to have a CorrelId field set, per the MsgId of the request (default).

Response has MsgID copied from MsgId
Expects the response message to have a MsgId field set, per the MsgId of the request.

Response has CorrelID copied from CorrelId
Expects the response message to have a CorrelId field set, per the CorrelId of the request.

Java EE resources

A number of Java EE resources are created when a WebSphere MQ binding is deployed to a Java EE environment.

Parameters

MQ Connection Factory
Used by clients to create a connection to the WebSphere MQ provider.

Response Connection Factory
Used by the SCA MQ runtime when the send destination is on a different Queue Manager than the receive destination.

Activation specification
An MQ JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

Destinations

- Send destination: where the request or outgoing message is sent (import); where the response message will be sent (export), if not superseded by the MQMD ReplyTo header field in the incoming message.
- Receive destination: where the response/request or incoming message should be placed.

WebSphere MQ headers:

WebSphere MQ headers incorporate certain conventions for conversion to the service component architecture (SCA) messages.

WebSphere MQ messages consist of a system header (the MQMD), zero or more other MQ headers (system or custom), and a message body. If multiple message headers exist in the message, the order of the headers is significant.

Each header contains information describing the structure of the following header. The MQMD describes the first header.

How MQ headers are parsed

An MQ Header data binding is used to parse MQ headers. The following headers are supported automatically:

- MQRFH
- MQRFH2
- MQCIH
- MQIIH

Headers that start with `MQH` are handled differently. Specific fields of the header are not parsed; they remain as unparsed bytes.

For other MQ headers, you can write custom MQ header data bindings to parse those headers.

How MQ headers are accessed

MQ headers can be accessed in the product in one of two ways:

- Through the service message object (SMO) in a mediation
- Through the ContextService API

MQ headers are represented internally with the SMO MQHeader element. MQHeader is a container of header data that extends MQControl but contains a value element of anyType. It contains the MQMD, MQControl (MQ message body control information), and a list of other MQ headers.

- MQMD represents the contents of the WebSphere MQ message description, except for information determining the structure and encoding of the body.
- MQControl contains information determining the structure and encoding of a message body.
- MQHeaders contain a list of MQHeader objects.

The MQ header chain is unwound so that, inside the SMO, each MQ header carries its own control information (CCSID, Encoding, and Format). Headers can be added or deleted easily, without altering other header data.

Setting fields in the MQMD

You can update the MQMD using the Context API or through the service message object (SMO) in a mediation. The following fields are automatically propagated to the outbound MQ message:

- Encoding
- CodedCharacterSet
- Format
- Report
- Expiry
- Feedback
- Priority
- Persistence
- CorrelId
- MsgFlags

Configure the MQ binding on an Import or Export to propagate the following properties to the outbound MQ message:

`MsgID`
: Set **Request Message ID** to `copy from SCA message`.

`MsgType`
: Clear the **Set message type to MQMT_DATAGRAM or MQMT_REQUEST for request-response operation** check box.

`ReplyToQ`
: Clear the **Override reply to queue of request message** check box.

`ReplyToQMgr`
: Clear the **Override reply to queue of request message** check box.

From version 7.0 onwards, context fields can be overridden using a custom property on the JNDI destination definition. Set the custom property MDCTX with value `SET_IDENTITY_CONTEXT` on the send destination to propagate the following fields to the outbound MQ message:

- UserIdentifier
- AppIdentityData

Set the custom property MDCTX with value `SET_ALL_CONTEXT` on the send destination to propagate the following properties to the outbound MQ message:

- UserIdentifier
- AppIdentityData
- PutApplType
- PutApplName
- ApplOriginData

Some fields are not propagated to the outbound MQ message. The following fields are overridden during the send of the message:

- BackoutCount
- AccountingToken
- PutDate
- PutTime
- Offset
- OriginalLength

Adding MQCIH statically in a WebSphere MQ binding:

WebSphere ESB supports adding MQCIH header information statically without using a mediation module.

There are various ways to add MQCIH header information to a message (for example, by using the Header Setter mediation primitive). It might be useful to add this header information statically, without the use of an additional mediation module. Static header information, including the CICS program name, the transaction ID, and other data format header details, can be defined and added as part of the WebSphere MQ binding.

WebSphere MQ, the MQ CICS Bridge, and CICS must be configured for MQCIH header information to be added statically.

You can use Integration Designer to configure the WebSphere MQ import with the static values that are required for the MQCIH header information.

When a message arrives and is processed by the WebSphere MQ import, a check is made to see if MQCIH header information is already present in the message. If the MQCIH is present, the static values defined in the WebSphere MQ import are used to override the corresponding dynamic values in the message. If the MQCIH is not present, one is created in the message and the static values defined in the WebSphere MQ import are added.

The static values defined in the WebSphere MQ import are specific to a method. You can specify different static MQCIH values for different methods within the same WebSphere MQ import.

This facility is not used to provide default values if the MQCIH does not contain specific header information because a static value defined in the WebSphere MQ import will override a corresponding value provided in the incoming message.

External clients:

WebSphere ESB can send messages to, or receive messages from, external clients using WebSphere MQ bindings.

An external client (for example, a Web portal or an enterprise information system) can send a message to an SCA component in the application by way of an export or it can be invoked by an SCA component in the application by way of an import.

The WebSphere MQ export binding deploys message driven beans (MDBs) to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications by way of the export binding.

WebSphere MQ imports bind to, and can deliver message to, external clients. This message might or might not demand a response from the external client.

More information on how to interact with external clients using WebSphere MQ can be found at the WebSphere MQ information center.

Troubleshooting WebSphere MQ bindings:

You can diagnose and fix faults and failure conditions that occur with WebSphere MQ bindings.

Primary failure conditions

The primary failure conditions of WebSphere MQ bindings are determined by transactional semantics, by WebSphere MQ configuration, or by reference to existing behavior in other components. The primary failure conditions include:

- Failure to connect to the WebSphere MQ queue manager or queue.

  A failure to connect to WebSphere MQ to receive messages will result in the MDB Listener Port failing to start. This condition will be logged in the WebSphere Application Server log. Persistent messages will remain on the WebSphere MQ queue until they are successfully retrieved (or expired by WebSphere MQ).

  A failure to connect to WebSphere MQ to send outbound messages will cause rollback of the transaction controlling the send.
- Failure to parse an inbound message or to construct an outbound message.

  A failure in the data binding causes rollback of the transaction controlling the work.
- Failure to send the outbound message.

  A failure to send a message causes rollback of the relevant transaction.
- Multiple or unexpected response messages.

  The import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

Misusage scenarios: comparison with WebSphere MQ JMS bindings

The WebSphere MQ import and export are principally designed to interoperate with native WebSphere MQ applications and expose the full content of the WebSphere MQ message body to mediations. The WebSphere MQ JMS binding, however, is designed to interoperate with JMS applications deployed against WebSphere MQ, which exposes messages according to the JMS message model.

The following scenarios should be built using the WebSphere MQ JMS binding, not the WebSphere MQ binding:

- Invoking a JMS message-driven bean (MDB) from an SCA module, where the MDB is deployed against the WebSphere MQ JMS provider. Use a WebSphere MQ JMS import.
- Allowing the SCA module to be called from a Java EE component servlet or EJB by way of JMS. Use a WebSphere MQ JMS export.
- Mediating the contents of a JMS MapMessage, in transit across WebSphere MQ. Use a WebSphere MQ JMS export and import in conjunction with the appropriate data binding.

There are situations in which the WebSphere MQ binding and WebSphere MQ JMS binding might be expected to interoperate. In particular, when you are bridging between Java EE and non-Java EE WebSphere MQ applications, use a WebSphere MQ export and WebSphere MQ JMS import (or vice versa) in conjunction with appropriate data bindings or mediation modules (or both).

Undelivered messages

If WebSphere MQ cannot deliver a message to its intended destination (because of configuration errors, for example), it sends the messages instead to a nominated dead-letter queue.

In doing so, it adds a dead-letter header to the start of the message body. This header contains the failure reasons, the original destination, and other information.

MQ-based SCA messages not appearing in the failed event manager

If SCA messages originated because of a WebSphere MQ interaction failure, you would expect to find these messages in the failed event manager. If these messages are not showing in the failed event manager, check that the underlying WebSphere MQ destination has a maximum failed deliveries value greater than 1. Setting this value to 2 or more allows interaction with the failed event manager during SCA invocations for the WebSphere MQ bindings.

MQ failed events are replayed to the wrong queue manager

When a predefined connection factory is to be used for outbound connections, the connection properties must match those defined in the activation specification used for inbound connections.

The predefined connection factory is used to create a connection when replaying a failed event and must therefore be configured to use the same queue manager from which the message was originally received.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a `DataBindingException`.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a `DataHandlerException` is reported to you as a `DataBindingException`.

When any runtime exception, including a `DataBindingException` exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a `DataHandlerException` is reported to you as a `DataBindingException`.

Limitations of bindings

The bindings have some limitations in their use that are listed here.

Limitations of the MQ binding:

The MQ binding has some limitations in its use that are listed here.

No publish-subscribe message distribution

The publish-subscribe method of distributing messages is not currently supported by the MQ binding though WMQ itself supports publish-subscribe. However, the MQ JMS binding does support this method of distribution.

Shared receive queues

Multiple WebSphere MQ export and import bindings expect that any messages present on their configured receive queue are intended for that export or import. Import and export bindings should be configured with the following considerations:

- Each MQ import must have a different receive queue because the MQ import binding assumes all messages on the receive queue are responses to requests

that it sent. If the receive queue is shared by more than one import, responses could be received by the wrong import and will fail to be correlated with the original request message.
- Each MQ export should have a different receive queue, because otherwise you cannot predict which export will get any particular request message.
- MQ imports and exports can point to the same send queue.

Limitations of the JMS, MQ JMS, and generic JMS bindings:

The JMS and MQ JMS bindings have some limitations.

Implications of generating default bindings

The limitations of using the JMS, MQ JMS, and generic JMS bindings are discussed in the following sections:
- Implications of generating default bindings
- Response correlation scheme
- Bidirectional support

When you generate a binding, several fields will be filled in for you as defaults, if you do not choose to enter the values yourself. For example, a connection factory name will be created for you. If you know that you will be putting your application on a server and accessing it remotely with a client, you should at binding creation time enter JNDI names rather than take the defaults since you will likely want to control these values through the administrative console at run time.

However, if you did accept the defaults and then find later that you cannot access your application from a remote client, you can use the administrative console to explicitly set the connection factory value. Locate the provider endpoints field in the connection factory settings and add a value such as <server_hostname>:7276 (if using the default port number).

Response correlation scheme

If you use the CorrelationId To CorrelationId response correlation scheme, used to correlate messages in a request-response operation, you must have a dynamic correlation ID in the message.

To create a dynamic correlation ID in a mediation module using the mediation flow editor, add an XSLT node before the import with the JMS binding. Open the XSLT mapping editor. The known service component architecture headers will be available in the target message. Drag and drop a field containing a unique ID in the source message onto the correlation ID in the JMS header in the target message.

Bidirectional support

Only ASCII characters are supported for Java Naming and Directory Interface (JNDI) names at runtime.

Shared receive queues

Multiple export and import bindings expect that any messages present on their configured receive queue are intended for that export or import. Import and export bindings should be configured with the following considerations:

- Each import binding must have a different receive queue because the import binding assumes all messages on the receive queue are responses to requests that it sent. If the receive queue is shared by more than one import, responses could be received by the wrong import and will fail to be correlated with the original request message.
- Each export should have a different receive queue, because otherwise you cannot predict which export will get any particular request message.
- Imports and exports can point to the same send queue.

Developing business process management solutions

This section discusses the fundamentals of the business process management (BPM) programming model. It introduces the Service Component Architecture (SCA) and discusses patterns related to business integration.

BPM is the discipline that enables companies to identify, consolidate, and optimize business processes. The objective is to improve productivity and maximize organizational effectiveness. Interest in BPM has become more acute as companies merge and consolidate, and as they grow a library of disparate information assets. These assets often lack consistency and coordination, thus giving rise to "islands of information."

BPM has strong links to Service-Oriented Architecture (SOA). Depending on the nature of the company and the extent of the integration needs, BPM poses different requirements for IT departments. Some projects deal with only a few aspects, whereas some larger projects encompass many of these requirements. Here are some of the most common aspects of BPM projects:

- **Application integration** is a common requirement. The complexity of application integration projects varies from simple cases, in which you need to ensure that few applications can share information, to more complex situations, in which transactions and data exchanges need to be reflected simultaneously on multiple back-end applications. Complex application integration often requires complex unit-of-work management as well as transformation and mapping.
- **Process automation** is another key aspect that ensures that activities performed by an individual or organization systematically trigger consequential activities elsewhere. This ensures the successful completion of the overall business process. For example, when a company hires an employee, payroll information has to be updated, appropriate actions need to be taken by the security department, the necessary tools need to be given to the employee, and so on. Some activities in a process might capture human input and interaction, whereas others might invoke scripts on back-end systems and other services in the environment.
- **Connectivity** is an abstract, yet critical, aspect both in a company and in terms of business partners. Connectivity refers to both the flow of information between organizations or companies and the ability to access distributed IT services.

Some of the technical challenges of business integration implementations can be summarized as follows:

- Dealing with different data formats and therefore not being able to perform efficient data transformation
- Dealing with different protocols and mechanisms for accessing IT services that have been developed using different technologies
- Orchestrating different IT services that may be geographically distributed or offered by different organizations
- Providing rules and mechanisms to classify and manage the services that are available (governance)

As such, BPM encompasses many of the themes and elements that are also common to SOA. IBM's vision of BPM builds on many of the same foundational concepts that are found in SOA. One of the immediate consequences of this vision is that BPM solutions require various products for their realization. IBM provides a portfolio of tools and runtime platforms to support all the various stages and operational aspects.

To paraphrase IBM's vision of BPM, it enables companies to define, create, merge, consolidate, and streamline business processes using applications that run on a SOA IT infrastructure. BPM work is truly role-based. At the macro level, it involves modeling, developing, governance, managing, and monitoring business process applications. With the help of proper tools and procedures, it enables you to automate business processes involving people and heterogeneous systems, both inside and outside the enterprise. One of the key aspects of BPM is the ability to optimize your business operations so that they are efficient, scalable, reliable, and flexible enough to handle change.

BPM requires development tools, runtime servers, monitoring tools, a service repository, toolkits, and process templates. Because there are so many aspects to BPM, you will find that you have to use more than one development tool to develop a solution. These tools enable integration developers to assemble complex business solutions. A server is a high-performance business engine or service container that runs complex applications. Management always wants to know who is doing what in the organization, and that is where monitoring tools come into play. As enterprises create these business processes or services, governance, classification, and storage of these services becomes critical. That function is served by a service repository. Specific toolkits to create specialized parts of the solution, such as connectors or adapters to existing systems, are often required.

BPM is not based on a single product. It involves almost everybody and all business aspects within an organization and across organizations. BPM encompasses many of the services and elements in the SOA reference architecture.

For more details about these concepts, along with programming examples, see:

- *WebSphere Business Integration Primer: Business Process Manager, BPEL, SCA, and SOA*, IBM Press, 2008.
- *Getting Started with IBM WebSphere Business Process Manager and IBM WebSphere Enterprise Service Bus Part 1: Development*, IBM Redbooks, SG24-7608-00, June 2008.
- *IBM Business Process Management Reviewer's Guide*, IBM Redpapers, REDP-4433-01, April 2009.

Business integration architecture and patterns

A typical business management project involves coordinating several different IT assets, potentially running on different platforms, and having been developed at

different times using different technologies. Being able to easily manipulate and exchange information with a diverse set of components is a major technical challenge. It is best addressed by the programming model used to develop business integration solutions.

This section introduces the Service Component Architecture (SCA) and discusses patterns related to business integration. Patterns seem to permeate our lives. Sewing patterns, think-and-learn patterns for children, home construction patterns, wood-carving patterns, flight patterns, wind patterns, practice patterns in medicine, customer buying patterns, workflow patterns, design patterns in computer science, and many more exist.

Patterns have proven successful in helping solution designers and developers. Therefore, it is not surprising that we now have business integration patterns and enterprise integration patterns. There is a wide array of patterns that are applicable to business integration, including patterns for request and response routing, channel patterns (such as publish/subscribe), and many more. Abstract patterns provide a template for resolving a certain category of problems, whereas concrete patterns provide more specific indications of how to implement a specific solution. This section focuses on patterns that deal with data and service invocation, which are at the foundation of the programming model of the IBM software strategy for WebSphere business integration.

Business integration scenarios

Enterprises have many different software systems that they use to run their business. In addition, they have their own ways of integrating these business components.

The two most prevalent business integration scenarios are as follows:

- **Integration broker**: In this scenario, the business integration solution acts as an intermediary located among various back-end applications. For example, you might have to ensure that when a customer places an order using the online order management application, the transaction updates relevant information in your Customer Relationship Management (CRM) back end. In this scenario, the integration solution must be able to capture and possibly transform the necessary information from the order management application and invoke the appropriate services in the CRM application.
- **Process automation**: In this scenario, the integration solution acts as the glue among different IT services that would otherwise be unrelated. For example, when a company hires an employee, the following sequence of actions must occur:
  - The information for the employee is added to the payroll system.
  - The employee must be granted physical access to the facilities, and a badge must be provided.
  - The company might have to assign a set of physical assets to the employee (office space, a computer, and so on).
  - The IT department must create a user profile for the employee and grant access to a series of applications.

    Automating this process is also a common use case in a business integration scenario. In this scenario, the solution implements an automated flow that is triggered by the addition of the employee to the payroll system. After, the flow triggers the other steps by creating work items for the people who are responsible for taking action or by calling the appropriate services.

In both scenarios, the integration solution must:

1. Work with disparate sources of information and different data formats, and be able to convert information between different formats
2. Be able to invoke various services, potentially using different invocation mechanisms and protocols.

Roles, products, and technical challenges

Successful business integration projects depend on the blending of specialized development roles, programming techniques, and tool suites.

Business integration projects require a few basic ingredients:

- A clear separation of roles in the development organization to promote specialization, which typically improves the quality of the individual components that are developed
- A common business object (BO) model that enables business information to be represented in a common logical model
- A programming model that strongly separates interfaces from implementations and that supports a generic service invocation mechanism that is totally independent of the implementation and that only involves dealing with interfaces
- An integrated set of tools and products that supports development roles and preserves their separation

The following sections elaborate on each of these ingredients.

Clear separation of roles

A business integration project requires people in four collaborative, but distinctly separate, roles:

- **Business analyst**: Business analysts are domain experts responsible for capturing the business aspects of a process and for creating a process model that adequately represents the process itself. Their focus is to optimize the financial performance of a process. Business analysts are not concerned with the technical aspects of implementing processes.
- **Component developer**: Component developers are responsible for implementing individual services and components. Their focus is the specific technology used for the implementation. This role requires a strong programming background.
- **Integration specialist**: This relatively new role describes the person who is responsible for assembling a set of existing components into a larger business integration solution. Integration developers do not need to know the technical details of each of the components and services they reuse and wire together. Ideally, integration developers are concerned only with understanding the interfaces of the services that they are assembling. Integration developers should rely on integration tools for the assembly process.
- **Solution deployer**: Solution deployers and administrators are concerned with making business integration solutions available to end users. Ideally, a solution deployer is primarily concerned with binding a solution to the physical resources ready for it to function (databases, queue managers, and so on) and not with having a deep understanding of the internals of a solution. The solution deployer's focus is quality of service (QoS).

A common business object model

As we discussed, the key aspects of a business integration project include the ability to coordinate the invocation of several components and the ability to handle the data exchange among those. In particular, different components can use different techniques to represent business items such as the data in an order, a customer's information, and so on. For example, you might have to integrate a Java application that uses entity Enterprise Java Beans (EJBs) to represent business items and a legacy application that organizes information in COBOL copybooks. Therefore, a platform that aims to simplify the creation of integration solutions should also provide a generic way to represent business items, irrespective of the techniques used by the back-end systems for data handling. This goal is achieved in WebSphere Business Process Manager and WebSphere Enterprise Service Bus thanks to the *business object framework*.

The business object framework enables developers to use XML Schemas to define the structure of business data and access and manipulate instances of these data structures (business objects) via XPath or Java code. The business object framework is based on the Service Data Object (SDO) standard.

The Service Component Architecture (SCA) programming model

The SCA programming model represents the foundation for any solution to be developed on WebSphere Business Process Manager and WebSphere Enterprise Service Bus. SCA provides a way for developers to encapsulate service implementations in reusable components. It enables you to define interfaces, implementations, and references in a way that is independent of which technology you use. This approach gives you the opportunity to bind the elements to whichever technology you choose. There is also an SCA client programming model that enables the invocation of those components. In particular, it enables runtime infrastructures based on Java to interact with non-Java runtimes. SCA uses business objects as the data items for service invocation.

Tools and products

IBM Integration Designer is the integrated development environment that has all the necessary tools to create and compose business integration solutions based on the technologies just mentioned. These solutions typically are deployed to the WebSphere Business Process Manager or, in some cases, to the WebSphere Enterprise Service Bus.

Chapter 7. Deploying modules directly to WebSphere ESB

Modules on WebSphere ESB are used to organize the components and services of applications. Deploying refers to the process of installing these modules to a test or production server where the applications can run and be further integrated into solutions.

Related information:

PDF documentation
WebSphere Enterprise Service Bus documentation (in PDF format)

Information roadmaps
Business Process Management information roadmaps on IBM developerWorks organize information about WebSphere Process Server, WebSphere ESB, and the other products in the portfolio.

IBM Education Assistant
Multimedia educational modules about WebSphere ESB, provided by IBM Education Assistant.

Technotes
WebSphere ESB Support > technote search for 7.5 “deploy” documents. Use the document type, product category, and search terms fields to find the information you need.

Overview
Overview tab, on product library Web page. Use this page to access announcements, data sheets, and other general library documents related to WebSphere ESB.

Overview of preparing and installing modules

Installing modules (also known as deploying) activates the modules in either a test environment or a production environment. This overview briefly describes the test and production environments and some of the steps involved in installing modules.

Note: The process for installing applications in a production environment is similar to the process described in “Developing and deploying applications” in the WebSphere Application Server Network Deployment information center. If you are unfamiliar with those topics, review those first.

Before installing a module to a production environment, always verify changes in a test environment. To install modules to a test environment, use Integration Designer (see the Integration Designer information center for more information). To install modules to a production environment, use WebSphere ESB.

This topic describes the concepts and tasks needed to prepare and install modules to a production environment. Other topics describe the files that house the objects that your module uses and help you move your module from your test environment into your production environment. It is important to understand these files and what they contain so you can be sure that you have correctly installed your modules.

© Copyright IBM Corp. 2007, 2011

Libraries and JAR files overview

Modules often use artifacts that are located in libraries, which are special projects in Integration Designer used for storing shared resources. At deployment time, Integration Designer libraries are transformed into utility JAR files and packaged in the applications to be run.

While developing a module, you might identify certain resources or components that could be used by other modules. These artifacts can be shared by using a library.

What is a library?

A library is a special project in Integration Designer that is used for the development, version management, and organization of shared resources, such as those resources that are typically shared between modules. Only a subset of artifact types can be created and stored in a library, including:

- Interfaces or Web services descriptors (files with a .wsdl extension)
- Business object XML schema definitions (files with an .xsd extension)
- Business object maps (files with a .map extension)
- Relationship and role definitions (files with a .rel and .rol extension)

At deployment time, these Integration Designer libraries are transformed into utility JAR files in the applications to be run.

When a module needs an artifact, the server locates the artifact from the EAR class path and loads the artifact, if it is not already loaded, into memory. Figure 307 on page 905 shows how an application contains components and related libraries.

Figure 307. Relationships among module, component, and library

What are JAR, RAR, and WAR files?

There are a number of files that can contain components of a module. These files are fully described in the Java Platform, Enterprise Edition specification. Details about JAR files can be found in the JAR specification.

In WebSphere ESB, a JAR file also contains an application, which is the assembled version of the module with all the supporting references and interfaces to any other service components used by the module. To completely install the application, you need this JAR file, any other dependent JAR, Web services archive (WAR), resource archive (RAR), staging libraries (Enterprise Java Beans - EJB) JAR files, and any other archives. You then create an installable EAR file using the **`serviceDeploy`** command.

Naming conventions for staging modules

Within the library, there are requirements for the names of the staging modules. These names are unique for a specific module. Name any other modules required to deploy the application so that conflicts with the staging module names do not occur. For a module named *myService*, the staging module names are:

- *myService*App
- *myService*Web

Note: The *myService*EJB and *myService*EJBClient staging modules no longer get created by **serviceDeploy**. However, those file names should not be used, because they could still be deleted by the **serviceDeploy** command.

Considerations when using libraries

Using libraries provides consistency of business objects and consistency of processing amongst modules because each calling module has its own copy of a specific component. To prevent inconsistencies and failures it is important to make sure that changes to components and business objects used by calling modules are coordinated with all of the calling modules. Update the calling modules by:

1. Copying the module and the latest copy of the libraries to the production server
2. Rebuilding the installable EAR file using the **serviceDeploy** command
3. Stopping the running application containing the calling module and reinstalling it
4. Restarting the application containing the calling module

EAR file overview

An EAR file is a critical piece in deploying a service application to a production server.

An enterprise archive (EAR) file is a compressed file that contains the libraries, enterprise beans, and JAR files that the application requires for deployment.

You create a JAR file when you export your application modules from IBM Integration Designer. Use this JAR file and any other artifact libraries or objects as input to the installation process. The **serviceDeploy** command creates an EAR file from the input files that contain the component descriptions and Java code that make up the application.

Preparing to deploy to a server

After developing and testing a module, you must export the module from a test system and bring it into a production environment for deployment. To install an application you also should be aware of the paths needed when exporting the module and any libraries the module requires.

Before you begin

Before beginning this task, you should have developed and tested your modules on a test server and resolved problems and performance issues.

Important: To prevent replacing an application or module already running in a deployment environment make sure the name of the module or application is unique from any already installed.

About this task

This task verifies that all of the necessary pieces of an application are available and packaged into the correct files to bring to the production server.

Note: You can also export an enterprise archive (EAR) file from Integration Designer and install that file directly into WebSphere ESB.

Important: If the services within a component use a database, install the application on a server directly connected to the database.

Procedure

1. Locate the folder that contains the components for the module you are to deploy.

   The component folder should be named *module-name* with a file in it named *module*.module, the base module.

2. Verify that all components contained in the module are in component subfolders beneath the module folder.

   For ease of use, name the subfolder similar to *module/component*.

3. Verify that all files that comprise each component are contained in the appropriate component subfolder and have a name similar to *component-file-name*.component.

   The component files contain the definitions for each individual component within the module.

4. Verify that all other components and artifacts are in the subfolders of the component that requires them.

   In this step you ensure that any references to artifacts required by a component are available. Names for components should not conflict with the names the **serviceDeploy** command uses for staging modules. See Naming conventions for staging modules.

5. Verify that a references file, *module*.references, exists in the module folder of step 1.

   The references file defines the references and the interfaces within the module.

6. Verify that a wires file, *module*.wires, exists in the component folder.

   The wires file completes the connections between the references and the interfaces within the module.

7. Verify that a manifest file, *module*.manifest, exists in the component folder.

   The manifest lists the module and all the components that comprise the module. It also contains a class path statement so that the **serviceDeploy** command can locate any other modules needed by the module.

8. Create a compressed file or a JAR file of the module as input to the **serviceDeploy** command that you will use to prepare the module for installation to the production server.

Example folder structure for MyValue module prior to deployment

The following example illustrates the directory structure for the module MyValueModule, which is made up of the components MyValue, CustomerInfo, and StockQuote.

```
MyValueModule
   MyValueModule.manifest
   MyValueModule.references
   MyValueModule.wiring
   MyValueClient.jsp
process/myvalue
   MyValue.component
   MyValue.java
   MyValueImpl.java
service/customerinfo
   CustomerInfo.component
```

```
   CustomerInfo.java
   Customer.java
   CustomerInfoImpl.java
service/stockquote
   StockQuote.component
   StockQuote.java
   StockQuoteAsynch.java
   StockQuoteCallback.java
   StockQuoteImpl.java
```

What to do next

Install the module onto the production systems as described in Installing a module on a production server.

Related information:

Troubleshooting a failed deployment
This topic describes the steps to take to determine the cause of a problem when deploying an application. It also presents some possible solutions.

Considerations for installing service applications on clusters

Installing a service application on a cluster places additional requirements on you. It is important that you keep these considerations in mind as you install any service applications on a cluster.

Clusters can provide many benefits to your processing environment by providing economies of scale to help you balance request workload across servers and provide a level of availability for clients of the applications. Consider the following before installing an application that contains services on a cluster:

- Will users of the application require the processing power and availability provided by clustering?

  If so, clustering is the correct solution. Clustering will increase the availability and capacity of your applications.
- Is the cluster correctly prepared for service applications?

  You must configure the cluster correctly before installing and starting the first application that contains a service. Failure to configure the cluster correctly prevents the applications from processing requests correctly.
- Does the cluster have a backup?

  You must install the application on the backup cluster also.

Cross-cluster modules

JNDI resources must not be shared across clusters. A cross-cluster module requires that each cluster have different JNDI resources. A scenario matching the following criteria will result in the log file indicating a `NameNotFoundException`:

- One module has a configured binding that generates JNDI resources.
- Another module is configured to use those generated JNDI resources.
- The modules are deployed in different clusters.

To resolve the problem, modify the module properties so that each module uses the JNDI resources in the same cluster as it.

Related information:

Considerations for Service Component Architecture (SCA) support in servers and clusters
Servers and clusters can support Service Component Architecture (SCA) applications, application destinations, or both.

Deploying a mediation module

You can deploy a mediation module, as generated by IBM Integration Designer, into a production environment by using these steps.

Before you begin

Before deploying a mediation module, assemble and test the module on a test server. After testing, export the mediation module and other files needed.

About this task

A mediation module provides functions that can change the format, content or target of service requests between applications and services. These functions can help you to integrate applications and services. Mediation modules are types of Service Component Architecture (SCA) modules.

To enable WebSphere ESB to use the functions provided by the mediation module you need to deploy the module. A mediation module is deployed to WebSphere ESB as a service application in the same way you would deploy enterprise applications, using an installable EAR (Enterprise ARchive) file.

During installation, you can configure the module enough to enable it to run in WebSphere ESB. After installation, you can configure the module further, start or stop the application, and otherwise manage its activity.

Note: If you need to deploy many application files, you may want to use ANT task, so you can define the deployment of multiple applications and have them run unattended on a server.

To deploy a mediation module as a service application, complete the following steps:

Procedure

1. Whether you have exported your mediation module as an EAR file, or a JAR file, copy the file containing your mediation module into the production environment.
2. Run the **serviceDeploy** command to create an installable EAR file.

   This step defines the module to the server in preparation for installing the application into production.

   a. Locate the JAR file that contains the module to deploy.
   b. Issue the **serviceDeploy** command using the JAR file from the previous step as input.
3. Install the EAR file onto an appropriate server or cluster. WebSphere ESB provides the same ways to install EAR files onto servers and clusters as provided by WebSphere Application Server. For information about how to use the administrative console to install the EAR file for a mediation module, see Installing an SCA module with the console.

4. When you are installing a mediation module there is one additional optional step you can perform. On the **Edit module properties** panel, you can edit the values of the properties in the module. If properties belong to a group they are displayed inside an expandable section; if they do not belong to a group you can view them immediately.

Results

The application is now active and work should flow through the module.

What to do next

Monitor the application to make sure the server is processing requests correctly.

Related concepts:

"Development and deployment of applications on WebSphere ESB" on page 24
Options for development and deployment of integrated applications on WebSphere ESB include working in the IBM Integration Designer development environment, working with Service Component Architecture APIs, and enabling the applications in a test or production server environment.

Installing versioned SCA modules in a production environment

You can deploy versioned Service Component Architecture (SCA) modules into the run time. Each version of a module exists alongside any other versions currently installed on the server or in the cell.

Before you begin

Make sure you perform the following tasks before installing a versioned SCA module into your production environment:

- In IBM Integration Designer, specify that the module is versioned and export it for command-line deployment. See Creating versioned modules and libraries for more information.
- Determine whether you want to co-deploy different versions of the module on a single server or whether you need to co-deploy multiple instances of the same versioned module on more than one cluster in the same cell.

About this task

To install versioned modules, perform the following steps.

Procedure

1. Run **serviceDeploy** against the versioned module you exported to generate an installable EAR file.

   ```
   serviceDeploy moduleName.zip
   ```

   The serviceDeploy command returns an installable EAR file whose name contains the version and, optionally, cell ID information.
2. Install the module using one of the following methods:
   - From within the administrative console, click SCA Modules and use the **Install** button on the SCA Modules page.
   - From within the administrative console, click Applications > Install New Application.

- Use the **AdminApp.install** wsadmin command.

3. If you want to install a versioned module on multiple servers or clusters in a cell, do the following for each module instance you require:
   a. Use the **createVersionedSCAModule** command to create an instance of the module.

      ```
      createVersionedSCAModule -archiveAbsolutePath input_archive_dir
      -workingDirectory working_dir -uniqueCellID cell_ID
      ```

   b. Install the resulting EAR file as described in Step 2 on page 910.

Results

You now have one or more versioned applications in your production environment. They can all be administered through the administrative console or through corresponding administrative commands.

Note: To preserve versioning information, the installation process automatically modifies the module name to ensure it is unique within the server or cell through the use of either the **serviceDeploy** or **createVersionedSCAModule** command. These commands add the version number, a unique cell ID, or both to the original module name.

moduleName_vversionValue_uniqueCellID

For example, if you followed the steps in this topic, deploying version 1.0.1 of the module billingProcess results in a module called billingProcess_v1_0_1 and an installed service application called billingProcess_v1_0_1App. If you also specify a unique cell ID (for example, Cell5), then the module is called billingProcess_v1_0_1_Cell5 and the installed service application is called billingProcess_v1_0_1_Cell5App.

Installing an SCA module with the console

Before you can start running an SCA module (mediation module), you must deploy it to a server or cluster. Deploying involves creating an installable enterprise archive (EAR) file and installing the EAR file onto the server or cluster.

Before you begin

If you have exported a module to a JAR file, use the **serviceDeploy** command to create an installable EAR file from the JAR file. For more information, see "Deploying a mediation module" on page 909.

Note: Versioned libraries are shared between applications for run time and management. A versioned library can be associated with an application or module using an installed optional package that declares the shared library in the application's manifest file. Modules that use a library have a dependency on a specific version of that library. An artifact in a versioned library that is used by multiple applications is managed as a single artifact. If the artifact is changed, it affects all dependents.

For details about versioned libraries, see the WebSphere Integration Developer information center.

About this task

You must install the EAR file onto a server or cluster before you can start running the SCA module. This is like installing any other enterprise application EAR file into WebSphere Application Server.

Instead of using the administrative console, you can use other methods to install the EAR file, such as the **`AdminApp.install`** or **`AdminApp.installinteractive`** command with the **`wsadmin`** tool.

Important: If, after you start performing the steps, you decide not to install the application you must click **Cancel**: do not simply move to another administrative console page.

Procedure

1. From the administrative console, click **Applications** > **New Application** in the console navigation pane. The first of two **Preparing for application installation** pages is displayed.
2. On the first **Preparing for application installation** page:
   a. Specify the full path name of the EAR file.
   b. Select whether to use default values or specify some of the values yourself:
      Prompt me only when additional information is required
      Displays only the module mapping step and other steps where you must specify information.
      Show me all installation options and parameters
      Displays all installation steps. To use **Generate default bindings**, which supplies default values for incomplete bindings, select this option.
   c. Click **Next**.
3. Complete the second **Preparing for application installation** page and specify the options in the remaining wizard steps
4. Optional: Edit the values of the properties in the module If you are installing a mediation module, you can edit the values of the properties in the module on the **Edit module properties** panel.

   If properties belong to a group they are displayed inside an expandable section; if they do not belong to a group you can view them immediately.

Results

You can now start the mediation module.

Related information:

Installing applications with the console
Description of installing the EAR file for a Java EE application, in the WebSphere Application Server information center.

Creating an installable EAR file using `serviceDeploy`

To install an application in the production environment, take the files copied to the production server and create an installable EAR file.

Before you begin

Before starting this task, you must have a JAR file that contains the module and services you are deploying to the server. See “Preparing to deploy to a server” for more information.

About this task

The **serviceDeploy** command takes a JAR file, any other dependent EAR, JAR, RAR, WAR and ZIP files and builds an EAR file that you can install on a server.

Procedure

1. Locate the JAR file that contains the module to deploy.
2. Issue the **serviceDeploy** command using the JAR file from the previous step as input.

   This step creates an EAR file.

   Note: Perform the following steps at an administrative console.
3. Select the EAR file to install in the administrative console of the server.
4. Click **Save** to install the EAR file.

Deploying applications using Apache Ant tasks

ANT tasks allow you to define the deployment of multiple applications to WebSphere ESB and have them run unattended on a server.

Before you begin

This task assumes the following:
- The applications being deployed have already been developed and tested.
- The applications are to be installed on the same server or servers.
- You have some knowledge of Apache Ant tasks.
- You understand the deployment process.

Information about developing and testing applications is located in the keyword Integration Designer information center.

The Generated API and SPI documentation reference section provides details of application programming interfaces. Apache Ant tasks are described in the package `com.ibm.websphere.ant.tasks`. For the purpose of this topic, the tasks of interest are **ServiceDeploy** and **InstallApplication**.

About this task

If you need to install multiple applications concurrently, develop an Apache Ant task before deployment. The Apache Ant task can then deploy and install the applications on the servers without your involvement in the process.

Procedure

1. Identify the applications to deploy.
2. Create a JAR file for each application.
3. Copy the JAR files to the target servers.
4. Create an Apache Ant task to run the **ServiceDeploy** command to create the EAR file for each server.

5. Create an Apache Ant task to run the **InstallApplication** command for each EAR file from step 4 on page 913 on the applicable servers.
6. Run the **ServiceDeploy** Apache Ant task to create the EAR file for the applications.
7. Run the **InstallApplication** Apache Ant task to install the EAR files from step 6.

Results

The applications are correctly deployed on the target servers.

Example of deploying an application unattended

This example of deploying an application unattended shows an Apache ANT task contained in a file `myBuildScript.xml`.

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

<project name="OwnTaskExample" default="main" basedir=".">
 <taskdef name="servicedeploy"
   classname="com.ibm.websphere.ant.tasks.ServiceDeployTask" />
 <target name="main" depends="main2">
  <servicedeploy scaModule="c:/synctest/SyncTargetJAR"
   ignoreErrors="true"
   outputApplication="c:/synctest/SyncTargetEAREAR"
   workingDirectory="c:/synctest"
   cleanStagingModules="true"/>
 </target>
</project>
```

This statement shows how to invoke the Apache Ant task.

```
${WAS}/bin/ws_ant -f myBuildScript.xml
```

Tip: Multiple applications can be deployed unattended by adding additional project statements into the file.

What to do next

Use the administrative console to verify that the newly-installed applications are started and processing the workflow correctly.

Related concepts:

"Development and deployment of applications on WebSphere ESB" on page 24
Options for development and deployment of integrated applications on WebSphere ESB include working in the IBM Integration Designer development environment, working with Service Component Architecture APIs, and enabling the applications in a test or production server environment.

Adapters and their installation

Adapters make it possible for your application to communicate with other components of your enterprise information system.

The process you use to install adapters is described in Configuring and using adapters.

Chapter 8. Securing WebSphere ESB and applications

Security of WebSphere ESB and applications depends on securing the runtime environment and securing applications.

Securing the WebSphere ESB runtime environment involves enabling administrative security, enabling application security, creating profiles with security, and restricting access to critical functions to selected users.

Securing an application includes authenticating users, implementing access control for operations and resources, and providing data integrity and privacy.

WebSphere ESB security is based on the WebSphere Application Server version 7.0 security.

For detailed information about security, see the WebSphere Application Server Network Deployment Information Center.

Security tasks can be broadly divided into those concerning the administration of security in the WebSphere ESB environment and those that are related to the applications running in WebSphere ESB. The security of the server environment is central to the security of applications, and therefore the two sides should not be thought of in isolation.

Securing the environment involves enabling administrative security, enabling application security, creating profiles with security, and restricting access to critical functions to selected users.

To secure an application, consider the following aspects:

- Authentication of users. A user or a process that invokes an application must be authenticated. With a single sign on, a user can provide authentication data once and then pass this authentication information to downstream components.
- Access control. The authenticated user must have permission to perform the operation.
- Data integrity and privacy. The data that is accessed by an application must be secured so that no unauthorized party can view or modify it in any way.

The rest of this section describes the security considerations at various stages of operation of the WebSphere ESB environment.

The following list provides an overview of the tasks you perform when securing WebSphere ESB.

1. Consider security when you install WebSphere ESB.
   a. Secure your environment before installation.
   b. Prepare the operating system for installation of WebSphere ESB.
   c. Prepare your environment after installation.
2. Ensure that security is turned on for your stand-alone or deployment environment installation.
   a. Ensure that Administrative security is turned on.
   b. Ensure that Application security is turned on.

© Copyright IBM Corp. 2007, 2011

c. If required, turn on Java 2 security.
d. Use the Security Configuration wizard in the administrative console to configure security options.
e. Set up a secure authentication mechanism and user account repository.
f. Assign user names and passwords to important business integration authentication aliases.
g. Assign users to appropriate administrative security roles.

3. Set up security for specific WebSphere ESB components. For example, use the Security Roles widget to set up role-based access control for timetables in the Business Calendars widget.
4. Secure the applications that you deploy to your process server environment.
   a. Develop your applications in Integration Designer using all appropriate security features.
   b. Deploy your applications to your WebSphere ESB environment.
   c. Assign users or groups to appropriate security roles to control access to the newly deployed application.
5. Maintain the security of your WebSphere ESB environment.

Getting started with security

Security is an integral consideration when you are planning to install WebSphere ESB, when you are developing and deploying applications, and in the day-to-day running of your process server.

Understanding elements of application security

Applications that run in WebSphere ESB are secured through a number of elements such as authentication, access control, data integrity, privacy, and identity propagation.

The following topics provide more information for application security in WebSphere ESB:

Access control

When authenticating a user for WebSphere ESB, it is important for security purposes that access to all operations not automatically be granted to that user. Allowing some users to perform certain operations, while denying access to those same operations for other users, is termed *access control*.

Access control can be arranged for components that you develop to make them secure. You provide access control for components by using service component architecture qualifiers at development time.

Some WebSphere ESB components, packaged as enterprise archive (EAR) files, secure their operation using Java EE role-based security. In contrast to code-based security, which secures the operation of components, role-based access control secures *resources*. For example, in the Business Calendars widget, you can specify the type of access that users have to individual timetables.

Security Roles widget

Use the Security Roles widget in Business Space to specify, for each timetable, the owner of the timetable as well as those who have writer and reader access to the timetable.

The following table shows the administrative roles and their default permissions:

Roles	Default permission
BPMAdmin	Primary administrative user
BPMRoleManager	All authenticated users

EAR files and associated roles

The Common Event Infrastructure is installed as part of WebSphere ESB.

Table 82. EAR files and associated roles in WebSphere ESB

Name of .ear file	Role	Default
BSpaceEAR_*nodeName_server*.ear	businessspaceusers	All Authenticated
BSpaceForms_*nodeName_server*.ear	WebFormUsers	All Authenticated
EventService.ear	eventAdministrator	All Authenticated
	eventConsumer	All Authenticated
	eventUpdater	All Authenticated
	eventCreator	All Authenticated
	catalogAdministrator	All Authenticated
	catalogReader	All Authenticated
mm.was_*nodeName_server*.ear	All Authenticated	All Authenticated
	everyone	Everyone
REST Services Gateway.ear	RestServicesUser	All Authenticated
REST Services Gateway Dmgr .ear	RestServicesUser	All Authenticated

Data integrity and privacy

The privacy and integrity of data when WebSphere ESB processes are invoked is critical to maintaining security.

Data privacy and data integrity are closely related concepts. For a more detailed discussion, refer to the WebSphere Application Server Network Deployment Information Center.

Privacy

Privacy means that an unauthorized user should not be able to intercept and read data.

Integrity

Integrity means that an unauthorized user should not be able to alter data.

Data integrity and privacy solutions provided in WebSphere ESB

WebSphere ESB supports two widely-used solutions for data privacy and integrity:

- Secure Sockets Layer (SSL) protocol: SSL uses a handshake to authenticate the end points and exchange information used to generate the session key that will be used by the end points for encryption and decryption. SSL is a synchronous

protocol and is suitable for point-to-point communication. SSL requires that the two end points maintain a connection with each other for the duration of the SSL session.

- WS-Security: This standard defines Simple Object Access Control (SOAP) extensions for securing SOAP messages. WS-Security adds support for authentication, integrity, and privacy for a single SOAP message. Unlike SSL, there is no handshake to establish a session key. This makes WS-Security suitable for securing messages in an asynchronous environment, such as SOAP over Java Message Service (JMS) or SOAP over Service Integration Bus (SIB). WS-Security deployment descriptors can be set in your applications before deployment.

In a business integration environment with multiple systems interacting with one another, it is likely that some of the communication will be asynchronous. Therefore, in most instances, WS-Security may be a more suitable solution.

Configuring a Web services Web client to use SSL

You can configure a Web services client to invoke a Web service using Secure Sockets Layer (SSL).

The details of how to configure your Web services Web client to use SSL are provided in this WebSphere Application Server technote. A more general discussion of securing Web services can be found in the WebSphere Application Server topic Securing applications at the transport level for Web services .

Single sign on

When a single sign on is used, a client is asked to provide the user name and password information only once. The provided identity then propagates throughout the system.

When a client request flows through multiple systems within the enterprise, the client must authenticate only once. This concept of identity propagation is solved using a single sign on method. The authenticated context is propagated to downstream systems, which can apply access control.

Either Tivoli Access Manager WebSEAL or Tivoli Access Manager plug-in for Web servers can be used as reverse proxy servers to provide access management and single sign on capability to WebSphere ESB resources. Details of how to configure these tools can be found in the WebSphere Application Server documentation.

Considerations for securing adapters

An adapter is the mechanism used by an application to communicate with an Enterprise Information System (EIS). The information exchanged between an application and an EIS can be highly sensitive and therefore important to ensure the security of this information transaction.

About this task

Two types of adapters are supported in WebSphere ESB: WebSphere Business Integration Adapters and WebSphere Adapters. The security of both types of adapters is discussed in the following topics:

Considerations for securing WebSphere Business Integration Adapters

When working with WebSphere Business Integration Adapters, there are a number of considerations for administering application security.

The following list provides security considerations when working with WebSphere Business Integration Adapters:

- WebSphere Business Integration Adapters consist of a collection of software, application program interfaces (APIs), and tools that enable applications to exchange business data through an integration broker. WebSphere Business Integration Adapters rely on JMS messaging which does not support security context propagation.
- For inbound communication from WebSphere Business Integration Adapters into WebSphere ESB, there is no authentication mechanism. The reliance on JMS messaging precludes security context propagation.
- The entry from an adapter to WebSphere ESB always employs a Service Component Architecture (SCA) export. The SCA export has to be wired to an SCA component, such as mediation, business process, SCA Java component, or Selector.
- The value for SecurityIdentity is a role, not a user. Nevertheless, when the EAR file is deployed to WebSphere ESB, you must provide a user name and password for the identity that is to be used. The use of SecurityIdentity prevents exceptions being thrown if a downstream component is secured and requires the client to have an authenticated identity.

  Note: The use of SecurityIdentity does not secure the communication between the adapter and the EIS.
- WebSphere Business Integration Adapters send data to WebSphere ESB as JMS messages over the service integration bus.

Considerations for securing WebSphere Adapters

When working with WebSphere Adapters, there are a number of considerations for administering application security.

The following list provides security considerations when working with WebSphere Adapters:

- WebSphere Adapters enable managed, bidirectional connectivity between an EIS and Java EE components supported by WebSphere ESB.
- For inbound communication from WebSphere Adapters into WebSphere ESB, there is no authentication mechanism. JCA also lacks inbound security support; therefore, WebSphere Adapters also have no authentication mechanism for inbound communication.
- The entry from an adapter to WebSphere ESB always employs a Service Component Architecture (SCA) export. The SCA export has to be wired to an SCA component, such as mediation, business process, SCA Java component, or Selector.
- The security solution is to define a runAs role on the component that is the target for the WebSphere Adapter export. This is done using the SCA qualifier SecurityIdentity during development (see the IBM Integration Designer Information Center for more information). When the component runs, it does so under the identity defined in the runAs role.
- The value for SecurityIdentity is a role, not a user. Nevertheless, when the EAR file is deployed to WebSphere ESB, you must provide a user name and

password for the identity that is to be used. The use of SecurityIdentity prevents exceptions being thrown if a downstream component is secured and requires the client to have an authenticated identity.

Note: The use of SecurityIdentity does not secure the communication between the adapter and the EIS.

- WebSphere Adapters reside in the JVM of the WebSphere ESB, and therefore only the communication between the adapter and the target EIS needs to be secured. The protocol between the adapter and the EIS is EIS-specific. The documentation of the EIS provides information about how to secure this link.

Creating end-to-end security

There are a number of potential end-to-end security scenarios, each involving differing security steps. Several typical scenarios, with the necessary security options, are presented in this topic.

Before you begin

These scenarios all assume that administrative security is enforced.

Procedure

1. Determine which of the examples provided in this section most closely match your security needs. In some instances, your needs might involve a combination of information from more than one of the scenarios.
2. Read the security information for the relevant scenarios and apply it to your security needs.

Example

The following examples illustrate common scenarios.

Example: Inbound Web service request

In this scenario, a Web service client invokes a component in WebSphere ESB. The request passes through several components in the WebSphere ESB environment before being passed to an EIS by an adapter.

You can authenticate the Web service client as an SSL client, using HTTP Basic authentication or using WS-Security authentication. When the client is authenticated, access control is applied based on the SecurityPermission qualifier. Between the client and the WebSphere ESB instance, you can secure the data integrity and privacy using SSL or WS-Security. SSL secures

the entire pipe, whereas with WS-Security, you can encrypt or digitally sign parts of the SOAP message. For Web services, WS-Security is the preferred standard.

Example: Outbound Web service request

In this scenario, the inbound request can be from an adapter, a Web service client, or an HTTP client. A component in WebSphere ESB (for example a mediation flow component) invokes an external Web service.

As for the inbound Web service request, you can authenticate with the external Web service as an SSL client, using HTTP Basic authentication or using WS-Security authentication. Use LTPACallBackHandler as the callback mechanism to extract the usernameToken from the current RunAs subject. Between WebSphere ESB and the target Web service, you can ensure data privacy and integrity using WS-Security.

Example: Web application - HTTP inbound request to WebSphere ESB

WebSphere ESB supports three types of authentication for HTTP:

- HTTP basic authentication
- HTTP forms-based authentication
- HTTPS SSL-based client authentication.

In addition, to protect your intranet from intruders, you can place the Web server in the demilitarized zone (DMZ) and the WebSphere ESB inside the inner firewall. In this example, WebSEAL is used as the reverse proxy, which performs the authentication. It has a trust association with WebSphere ESB behind the firewall and can forward authenticated requests.

Defining RunAs roles user assignments for system applications

The bpmMpdifyMapRunAsRoles script provides a way to define the RunAs roles user assignments for the system applications that are shipped with the WebSphere ESB product.

About this task

For more information on RunAs roles, see Assigning users to RunAs.

To define the RunAs roles user assignments, run the `bpmMpdifyMapRunAsRoles` script:

- Standalone environment

  ```
  wsadmin -username username -password password -f bpmMpdifyMapRunAsRoles.py -usr user -pwd pwd -nodeName node -serverName server
  ```

  where

  `username` is the admin user

  `password` is the admin password

  `user` is the user to be set for the run as roles

  `pwd` is the pwd to be set for the run as roles

  `node` is the name of the node

  `server` is the name of the server
- Clustered environment

  ```
  wsadmin -username username -password password -f bpmMpdifyMapRunAsRoles.py -user user -pwd pwd -clusterName cluster
  ```

  where

  `username` is the admin user

  `password` is the admin password

  `user` is the user to be set for the run as roles

  `pwd` is the pwd to be set for the run as roles

  `cluster` is the name of the cluster

Configuring administrative and application security

The first step in securing your WebSphere ESB environment and your applications is to make sure that administrative security is enabled.

Before you begin

- Install WebSphere ESB and verify the installation before performing these tasks.
- Open the administrative console for the profile that you want to secure. Log in to the console using an account that has administrator privileges, for example, the default administrator account that was specified during installation.

About this task

Using the administrative console, you can enable administrative security, application security, and Java 2 security.

- *Administrative security* is enabled by default, and applies to every server within the security domain.

  Administrative security determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of

which act as defaults. Proper planning is required because incorrectly enabling administrative security can lock you out of the administrative console or cause the server to end abnormally.

Administrative security can be thought of as a "big switch" that activates a wide variety of security settings for WebSphere ESB. Values for these settings can be specified, but they will not take effect until administrative security is activated. The settings include the authentication of users, the use of Secure Sockets Layer (SSL), and the choice of user account repository. In particular, application security, including authentication and role-based authorization, is not enforced unless administrative security is active.

- *Application security* is also enabled by default, and is in effect only when administrative security is enabled.

  Application security enables security for the applications in your environment. This type of security provides application isolation and requirements for authenticating application users.

- *Java 2 security* is disabled by default.

  Java 2 security provides a policy-based, fine-grained access control mechanism that increases overall system integrity by checking for permissions before allowing access to certain protected system resources. Java 2 security guards access to system resources such as file I/O, sockets, and properties. It also guards access to web resources such as servlets, JavaServer Pages (JSP) files, and Enterprise JavaBeans (EJB) methods.

  Because Java 2 security is relatively new, many existing applications, or even new applications, might not be prepared for the fine-grained access control programming model that this form of security is capable of enforcing. Administrators need to understand the possible consequences of enabling Java 2 security if applications are not prepared for it. Java 2 Security places some new requirements on application developers and administrators.

  Attention: Fix packs that include updates to the Software Development Kit (SDK) might overwrite unrestricted policy files. Back up unrestricted policy files before you apply a fix pack and reapply these files after the fix pack is applied.

Procedure

1. Open the administrative security page in the administrative console.

   Expand **Security** and click **Global security**.

2. Confirm that **Enable administrative security** is selected. If not, select this option.
3. Confirm that **Enable application security** is selected. If not, select this option.
4. Optional: Enable Java 2 security, if required.

   Although Java 2 security is supported, it is disabled by default. Select **Use Java 2 security to restrict application access to local resources** to enforce Java 2 security permission checking.

   When you enable Java 2 security, an application that requires more Java 2 security permissions than are granted in the default policy might fail to run properly until the required permissions are granted in either the `app.policy` file or the `was.policy` file of the application. Access Control exceptions are generated by applications that do not have all the required permissions. For more information about Java 2 security, see the topic on Configuring Java 2 security policy files in the WebSphere Application Server Information Center. A related link is provided.

Note: Updates to the `app.policy` file apply only to the enterprise applications on the node to which the `app.policy` file belongs.

a. Optional: Select **Warn if applications are granted custom permissions**. The `filter.policy` file contains a list of permissions that an application should not have according to the J2EE 1.4 Specification. If an application is installed with a permission specified in this policy file and this option is enabled, a warning is issued. The default is enabled.

b. Optional: Select **Restrict access to resource authentication data**. Enable this option if you need to restrict application access to sensitive Java Connector Architecture (JCA) mapping authentication data.

5. If you made changes to the security settings, perform these additional steps.

   a. Click **Apply**.

   b. Click **Save**.

   c. If necessary, stop and restart the server.

What to do next

You must confirm that administrative security is enabled for each profile that you create.

Configuring administrative security

During installation, all components default to the primary administrative credentials that you provide. These default values provide basic security, but to enhance the security of your installation, configure the various components of WebSphere ESB with appropriate security identities.

About this task

When you create an WebSphere ESB profile and keep **Enable administrative security** selected, you are prompted for a user name. This identity is used as a default for all underlying components. Configure these identities after you create profiles to further enhance your security

Several components of WebSphere ESB use authentication aliases. These aliases are used to authenticate the runtime component for access to databases and messaging engines. You can modify these aliases on the Business Integration Security page of the administrative console.

Starting and stopping the server

When administrative security is enabled, you must provide the appropriate user name and password to stop the server. The server will start without authentication, however, that authentication is required to access the administrative console.

Before you begin

Administrative security must be enabled.

Avoid trouble: Vista Windows 7 If User Account Control (UAC) is enabled on some levels, the application server must be started with Administrator privileges if you are using a command prompt. Start the application server from a command prompt window that is launched by performing the following actions:

- Right-click a command prompt shortcut.
- Click **Run As Administrator**.

- When you open the command prompt window as Administrator, an operating-system dialog appears that asks you if you want to continue. Click **Continue** to proceed.

Procedure

1. Start the server.

   The following table describes the options for starting the server.

Start the server	Details
From the First Steps user interface	Click **Start the server**.
From a command line	Enter: • Windows **On Windows platforms:** `startserver` *servername* • Linux UNIX **On Linux and UNIX platforms:** `startserver.sh` *servername*

   Note: You are not required to provide a user name and password to start the server. However, you will need to authenticate yourself if you try to launch the administrative console or perform any other administrative task.
   The server starts or an error message is returned.

2. Stop the server.

   The following table describes the options for stopping the server.

Stop the server	Details
From the First Steps user interface	Click **Stop the server** and provide a valid user name and password when prompted. The user name you provide must be in either the operator or administrator role.
From a command line	Enter: • Windows **On Windows platforms:** `stopserver` *servername* `-profileName` *ProfileName* `-username` *username* `-password` *password* • Linux UNIX **On Linux and UNIX platforms:** `stopserver.sh` *servername* `-profileName` *ProfileName* `-username` *username* `-password` *password*

   Note: You are required to provide a user name and password to stop the server.
   If the user name and password you provide are members of the operator or administrator roles, the server will stop.

3. Check that the server stopped successfully

   The following table describes the options for verifying that the server stopped correctly.

Check that the server stopped successfully	Details
From the user interface	The First Steps output window details the results of your request.

Check that the server stopped successfully	Details
From a command line	The outcome of your request is displayed in the command window from which the request was made.

Administrative security roles

Several administrative security roles are provided as part of the WebSphere ESB installation.

There are eight roles provided as part of the administrative console. These roles grant permission to ranges of functionality on the administrative console. When administrative security is enabled, a user must be mapped to one of these roles in order to access the administrative console.

The first user to log in to the server after installation is added to the administrator role.

Table 83. Administrative security roles

Administrative security role	Description
Monitor	A member of the monitor role can view the WebSphere ESB configuration and the current state of the server.
Configurator	A member of the configurator role can edit the WebSphere ESB configuration.
Operator	A member of the operator role has monitor privileges, plus the ability to modify the runtime state (that is, start and stop the server).
Administrator	The administrator role is a combination of configurator and operator roles plus additional privileges granted solely to the administrator role. Examples include: • Modifying the server user ID and password • Mapping users and groups to the administrator role The administrator also has the permission required to access sensitive information, such as: • Lightweight Third Party Authentication (LTPA) passwords • Keys
ISC Admins	This role is available only for administrative console users and not for **wsadmin** users. Users who are granted this role have administrator privileges for managing users and groups in the federated repositories. For example, a user of the ISC Admins role can complete the following tasks: • Create, update, or delete users in the federated repositories configuration • Create, update, or delete groups in the federated repositories configuration
Deployer	Users who are granted this role can perform both configuration actions and runtime operations on applications.

Table 83. Administrative security roles (continued)

Administrative security role	Description
Admin Security Manager	Only users who are granted this role can map users to administrative roles. Also, when fine-grained administrative security is used, only users who are granted this role can manage authorization groups.
Auditor	Users who are granted this role can view and modify the configuration settings for the security auditing subsystem. **Note:** The auditor role includes the monitor role. This allows the auditor to view but not change the rest of the security configuration.

See Administrative roles in the WebSphere Application Server information center for more information.

The server ID that is specified when you enable administrative security is automatically mapped to the administrator role. Users or groups can be added to and removed from the administrative roles at any time through the WebSphere ESB administrative console. However, a server restart is required for the changes to take effect.

Tip: Map a group or groups, rather than specific users, to administrative roles because it is more flexible and easier to administer. By mapping a group to an administrative role, adding or removing users to or from the group occurs outside of WebSphere ESB and does not require a server restart for the change to take effect.

The failed event manager can be operated by any user granted either the administrator or the operator role.

Selectors can be configured by any user granted either the administrator or the configurator role

In addition to mapping users or groups, a special-subject can also be mapped to the administrative roles. A special-subject is a generalization of a particular class of users.

- The **AllAuthenticated** special-subject means that the access check of the administrative role ensures that the user making the request is at least authenticated.
- The **Everyone** special-subject means that anyone, authenticated or not, can perform the action, as if security were not enabled.

Configuring application security

Applications deployed to your WebSphere ESB instance require security to be built into them which will later be applied at runtime.

About this task

The applications that you host in your WebSphere ESB environment perform many business critical functions that require security. Some applications will access, transfer, or alter sensitive information (for example, payroll information or credit card details). Others will perform billing or inventory management. The security of these applications is vitally important.

Secure your applications by doing the following:

Procedure

1. Ensure that administrative security is enabled.
2. Ensure that application security is enabled.
   a. On the administrative console, expand **Security** and click **Global security**.
   b. Select **Enable application security** so that WebSphere ESB will require authentication from users who try to access a secured application.
3. Develop your applications in Integration Designer using all appropriate security features.
4. Deploy your applications to your WebSphere ESB environment, assigning users or groups to appropriate security roles.
5. Maintain the security of your WebSphere ESB environment.

Enabling security for additional Process Server components

When a profile is created, default values are used for security credentials. After the profile is created, the security settings can be configured using the administrative console.

About this task

When you create a profile, the following components of WebSphere ESB take the administrator user identity as a default:

- Service component architecture (SCA)
- Business Process Choreographer
- Common Event Infrastructure (CEI)

The identities associated with these components are used to create authentication aliases that are required when security is enabled. It is important to change these identities to appropriate users from your account repository.

Procedure

To create and configure the security settings for a profile, perform the following steps.

1. To display the Business Integration Security page, expand **Security** and click **Business Integration Security**.
2. For each of the Service Component Architecture, Business Process Choreographer, and Common Event Infrastructure authentication aliases, provide an appropriate user name and password to use as an authentication alias.
   a. To select the alias that you want to change, click its name in the **Alias** column. In some cases, the **Alias** column might not provide a link, in which case you select the check box in the **Select** column corresponding to the alias that you want to edit, and click **Edit**.
   b. On the next page, provide the user name and password that is to be used as the authentication alias for this component.

      Note: The credentials that you provide must exist in the user account repository that you are using.
   c. Click **OK**.

Deploying (installing) secure applications

Deploying applications that have security constraints (secured applications) is similar to deploying applications with no security constraints. The only difference is that you might need to assign users and groups to roles for a secured application, which requires that you have the correct active user registry. If you are installing a secured application, roles would have been defined in the application. If delegation was required in the application, RunAs roles also are defined and a valid user name and password must be provided.

Before you begin

Before you perform this task, verify that you have designed, developed, and assembled an application with all the relevant security configurations. For more information about these tasks, see the Integration Designer information center. In this context, deploying and installing an application are considered the same task.

About this task

One of the required steps to deploy secured applications is to assign users and groups to the roles that were defined when the application was constructed. This task is completed as part of the step entitled, “Map security roles to users and groups”. If an assembly tool was employed, this assignment might have been completed in advance. In that case, you can confirm the mapping by completing this step. You can add new users and groups and modify existing information during this step.

If a RunAs role has been defined in the application, the application will invoke methods using an identity setup during deployment. Use the RunAs role to specify the identity under which the downstream invocations are made. For example, if the RunAs role is assigned user “bob”, and the client, “alice”, is invoking a servlet (with delegation set) that calls the enterprise beans, the method on the enterprise beans is invoked with “bob” as the identity.

As part of the deployment process, one of the steps is to assign or modify users to the RunAs roles. This step is entitled, “Map RunAs roles to users”. Use this step to assign new users or modify existing users to RunAs roles when the delegation policy is set to SpecifiedIdentity.

The steps described in the Procedure section are common for both installing an application and modifying an existing application. If the application contains roles, you see the **Map security roles to users and groups** link during application installation and also during managing applications, as a link in the Additional properties section.

Procedure

1. Go to the installed application that requires users to be mapped to the roles.

   Complete the steps that are required for installing applications before the step entitled, "Map security roles to users and groups".
2. Assign users and groups to roles.
3. Map users to RunAs roles if RunAs roles exist in the application.
4. Click **Correct use of System Identity** to specify RunAs roles, if needed.

   Complete this action if the application has delegation set to use system identity, which is applicable to enterprise beans only. System identity uses the

WebSphere ESB security server ID to invoke downstream methods. Use this ID with caution because this ID has more privileges than other identities in accessing WebSphere ESB internal methods. This task is provided to make sure that the deployer is aware that the methods listed in the page have system identity set up for delegation and to correct them if necessary. If no changes are necessary, skip this task.

5. Complete the remaining non-security related steps to finish installing and deploying the application.

What to do next

After a secured application is deployed, verify that you can access the resources in the application with the correct credentials. For example, if your application has a protected Web module, make sure only the users that you assigned to the roles can use the application.

Assigning users to roles

A secured application uses one or both of the security qualifiers securityPermission and securityIdentity. When these qualifiers are present, there are additional steps that must be taken at deployment time in order that the application and its security features work correctly.

Before you begin

This task assumes that you have a secured application ready to deploy as an EAR file into WebSphere ESB.

About this task

Applications implement interfaces that have methods. You can secure an interface or a method with the Service Component Architecture (SCA) qualifier securityPermission. When you invoke this qualifier, you specify a role (for example, “supervisors”) that has permission to invoke the secured method. When you deploy the application you have the opportunity to assign users to the specified role.

The securityIdentity qualifier is equivalent to the RunAs role used for delegations in WebSphere Application Server. The value associated with this qualifier is a role. During deployment, the role is mapped to an identity. Invocation of a component secured with securityIdentity takes the specified identity, regardless of the identity of the user who is invoking the application.

Procedure

1. Follow the instructions for deploying an application into WebSphere ESB. See Deploying a mediation module for more details.
2. Associate the correct users with the roles.

Security qualifier	Action to take
Security Permission	Assign a user or users to the role specified. There are four choices: • Everyone - equivalent to no security. • All authenticated - every authenticated user is a member of the role. • Mapped User - Individual users are added to the role. • Mapped Groups - Groups of users are added to the role. The most flexible choice is Mapped Groups, because users can be added to the group and thus gain access to the application without restarting the server.
Security Identity	Provide a valid user name and password for the identity to which the role is mapped.

Securing access to timetables in the Business Calendars widget

The Security Roles widget provides you with the ability to secure access to individual timetables in the Business Calendars widget. You use the Security Roles widget to assign roles to the members of an organization. It is these roles that determine the level of access to the timetables.

The Security Roles widget, which you use to administer role-based access control for the Business Calendars widget, is located in Business Space powered by WebSphere.

This role-based access is based on XACML (eXtensible Access Control Markup Language), an open standard.

What are the advantages of using the Security Roles widget role-based access control in the Business Calendars widget?

- You can control access to a specific instance of a timetable.

  For example, you can specify that a user has access only to the user's own timetable and that the user does not have the ability to look at or change anyone else's timetable.
- Controlling access is done at the role level, instead of the individual user level.

  You map members to roles. It is the role that defines the permission members have to the specific instance of the resource.

Using the Security Roles widget, you can assign a user or group to the system roles. You can also assign a user or group to the component roles associated with timetables.

Roles associated with a timetable

When a timetable is installed, three roles are created for that timetable–Owner, Writer, and Reader. These roles are known as component-specific roles.

How would these roles be used? Consider the case of a holiday timetable used in an organization. You want all employees to have access to the timetable, but you want to limit the number of employees who can update the timetable.

When the Holiday timetable is installed, the following roles are created:

- HolidayOwner

  Members assigned to this role can read the Holiday timetable and can also write to it. For example, if the company decided to add an extra holiday, a member with the HolidayOwner role would be able to make the change.

  Members of this role can also assign members to the HolidayWriter and HolidayReader role. For example, the HolidayOwner might decide to add a senior manager to the HolidayWriter role.
- HolidayWriter

  Members assigned to this role can read the Holiday timetable and can also write to it. As in the case of the HolidayOwner, members of the HolidayWriter role could add the extra holiday.
- HolidayReader

  Members assigned to this role can read the Holiday timetable but cannot write to it.

You might assign the HolidayOwner role to the Human Resources manager, the HolidayWriter role to the Human Resources Specialists group, and the HolidayReader role to the employee group, as shown in the following figure:

Figure 308. Example of roles assigned to a timetable

When you deploy a timetable, the three roles–Owner, Writer, and Reader–are created. Permission for all roles is set initially to **All Authenticated**. Make sure that you change this designation to assign the members of the organization to the correct roles.

Note: You can change the membership of a role (for example, you can remove a member from the reader role), but you cannot change the name of a role, add or delete a role, or change the permissions associated with a role. The permissions are set as follows:

- Members of the Owner role can read and write to the timetable and can assign other members to the Writer and Reader roles.
- Members of the Writer role can read and write to the timetable.
- Members of the Reader role can read the timetable.

In the Security Roles widget, these timetable-related roles are also known as *module roles*.

System roles for the Security Roles widget

The BPMAdmin and BPMRoleManager roles are automatically created when you enable security after installing WebSphere ESB.

- BPMAdmin

  BPMAdmin has the authority to add members to or remove members from the BPMRoleManager role. For example, if the person performing the BPMRoleManager role leaves the organization, only BPMAdmin can assign another member to that role.

  BPMAdmin is initially assigned to one member–the primary administrative user. Change this assignment to another member as soon as you restart the server after installation or upgrade.
- BPMRoleManager

  BPMRoleManager has the authority to add members to or remove members from the three timetable-related roles-Owner, Writer, and Reader. For example, if a Holiday timetable is created, the BPMRoleManager assigns members to the HolidayOwner, HolidayWriter, and HolidayReader roles.

  BPMRoleManager is initially assigned to one member–the primary administrative user. Change this assignment to another member as soon as you restart the server after installation or upgrade.

Assigning component roles

Each timetable in the Business Calendars widget has three component roles–Owner, Writer, and Reader–associated with it. You use the Security Roles widget to assign users or groups to these roles.

Before you begin

Make sure the Security Roles widget is displayed.

About this task

The BPMRoleManager can assign users or groups to component roles.

The Owner of a timetable can also assign users or groups to the Owner, Writer, or Reader role for that timetable.

Procedure

1. To assign individual members to a module role, complete the following steps:
   a. From the **Module** list, select a timetable.

b. For a role (for example, the Writer role for the timetable), click the name of the role.
c. On the right side of the page, click **Add**.
d. Type a name (or part of a name) in the **Users or groups to search for** field.
e. To restrict the number of users or groups returned based on the search criteria, change the value in the **Maximum result** field. Set this value to `0` to return the entire result set.
f. Click **Search**.
g. From the list that is displayed, select one or more users or groups and click **OK**.
h. When you have assigned all members, click **Save**.

2. To assign all members to a module role, complete the following steps:
   a. From the **Module** list, select a timetable.
   b. For a role (for example, the Reader role for the timetable), click the name of the role.
   c. Select **All authenticated**.
   d. Click **Save**.

Setting up security for Business Space

If you are using Business Space powered by WebSphere with your environment, you must consider security options for how your team will work with artifacts in Business Space. If you want to turn on security for Business Space, set up application security and designate a user repository. To define Business Space administrators, assign a superuser role.

About this task

For best results, enable security before you configure Business Space. If you enable security later, use the administrative console Global security administration page, to enable both administrative security and application security. On the same administrative console page, you also can designate a user account repository, including changing from the default federated repositories option to another user repository. To designate which users can perform Business Space administrator actions in the Business Space environment, assign the Business Space superuser role. Other security configuration may be needed for your specific environment.

Important: By default, the Ajax proxy configuration used with Business Space widgets does not restrict access to any IP addresses. For convenience, the Ajax proxy is configured by default to be open, which is not secure for production scenarios. To configure the Ajax proxy so that it displays only content from selected sites or blocks content from selected sites, follow the steps at Blocking IP addresses using the Business Space Ajax proxy.

Enabling security for Business Space

If you expect to use a secured environment, enable security before you configure Business Space. However, if needed, you can enable security manually later. To turn on security for Business Space you must enable both application security and administrative security.

Before you begin

Before you complete this task, you must have completed the following tasks:

- Check that your user ID is registered in the user registry for your product.

About this task

Business Space is preconfigured to ensure authentication and authorization of access. Users are prompted to authenticate when accessing Business Space URLs. Unauthenticated users are redirected to a login page.

Business Space is configured to be accessed by HTTPS by default. If you prefer HTTP because your system is already behind a firewall, you can switch to HTTP by running the `configBSpaceTransport.py` script. The `configBSpaceTransport.py` script has parameters to switch to either HTTP or HTTPS if you want to change from a previous setting. See Designating HTTP or HTTPS settings for Business Space.

To enable authenticated access to Business Space, you must have a user registry configured and application security enabled. Authorization to spaces and page content in Business Space is handled internally to Business Space as part of managing spaces.

Procedure

1. For complete instructions on security, see the security documentation for your product.
2. For the Business Space application, on the Global security administrative console page, select both **Enable administrative security** and **Enable application security**.
3. If you want to enable or remove security after you have configured Business Space with your profile, you must modify the `noSecurityAdminInternalUserOnly` property in the `ConfigServices.properties` file.

   The `noSecurityAdminInternalUserOnly` property specifies the Business Space administrator ID when security is disabled. By default, Business Space configuration will set the property to **BPMAdministrator** if security is disabled. When security is enabled, by default this property is set to the application server admin ID. If you want to enable or remove security after you have configured Business Space, use the application server admin ID.

   a. Modify the `ConfigServices.properties` file `noSecurityAdminInternalUserOnly` property to set it to the application server admin ID. The `ConfigServices.properties` file is located at *profile_root*`\BusinessSpace\`*node_name*`\`*server_name*`\mm.runtime.prof\config\ConfigService.properties` for a stand-alone server or *deployment_manager_profile_root*`\BusinessSpace\`*cluster_name*`\mm.runtime.prof\config\ConfigService.properties` for a cluster.

   b. Run the **updatePropertyConfig** command using the wsadmin scripting client.

      - For a stand-alone server:

        The following example uses Jython:

        ```
        AdminTask.updatePropertyConfig('[-serverName server_name -nodeName node_name
        -propertyFileName "profile_root\BusinessSpace\node_name\server_name
        \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"]')
        AdminConfig.save()
        ```

The following example uses Jacl:

```
$AdminTask updatePropertyConfig {-serverName server_name -nodeName node_name
 -propertyFileName "profile_root\BusinessSpace\node_name\server_name
\mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"}
$AdminConfig save
```

- For a cluster:

  The following example uses Jython:

```
AdminTask.updatePropertyConfig('[-clusterName cluster_name -propertyFileName
 "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
config\ConfigService.properties" -prefix "Mashups_"]')
AdminConfig.save()
```

  The following example uses Jacl:

```
$AdminTask updatePropertyConfig {-clusterName cluster_name -propertyFileName
 "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
config\ConfigService.properties" -prefix "Mashups_"}
$AdminConfig save
```

c. Restart the server.

d. Log into Business Space and reassign the owners of the default spaces to the new administrator ID.

What to do next

- After the administrative security and application security are turned on, you receive a prompt for a user ID and password when you log on to Business Space. You must use a valid user ID and password from the selected user registry in order to log on. After you turn on administrative security, whenever you return to the administrative console, you must log on with the user ID that has administrative authority.
- If you want to change the user account repository from the default for your product profile, follow the steps in Selecting the user account repository for Business Space.
- If you have a cross-cell environment where Business Space is remote from where your product is running, and the nodes are not in the same cell, set up single-sign-on (SSO) and Secure Sockets Layer (SSL) certificates. Follow the instructions in Setting up SSO and SSL for Business Space.
- To designate who can perform Business Space administrator actions in the Business Space environment, see Assigning the Business Space superuser role.

Selecting the user repository for Business Space

The federated repositories option is the default user account repository option for profiles. You can change the type of user account repository if needed for your environment.

Before you begin

Before you complete this task, you must have completed the following tasks:

- Enable application security and administrative security. See “Enabling security for Business Space” on page 934.
- Check that your user ID is registered in the user registry for your product.

About this task

To enable authenticated access to Business Space, you must have a user registry configured and application security enabled. For information about application security, see “Enabling security for Business Space” on page 934.

Considerations for using a user account registry with Business Space:

- Based on the type of LDAP configuration that you are using, your settings can impact your ability to access Business Space correctly. Make sure that the user filters, the group filters, and mapping settings are configured properly. For more information, see Configuring Lightweight Directory Access Protocol search filters in the WebSphere Application Server documentation.
- Based on the type of federated repository configuration that you are using, your settings can affect your ability to access Business Space correctly. Make sure that the realms are configured properly. For more information, see Managing the realm in a federated repository configuration in the WebSphere Application Server documentation.
- The LDAP security is set up by default to use the login property `uid` (user ID) for searching in Business Space. If your LDAP security is changed to use another unique LDAP field, such as `mail` (email address) for the login property, then you must modify the `userIdKey` property in the `ConfigServices.properties` file in order for searching to work in Business Space. Follow step 3 below.
- If you are using a Microsoft SQL Server database and the **Standalone LDAP** registry, make sure that the user distinguished name (user DN) does not exceed 450 characters. If any of the user DN entries exceed 450 characters, you must designate the **Federated repositories** option for the user account repository.
- If you are using **Federated repositories**, you have additional capabilities in your widgets and framework, such as enhanced search capabilities. When searching for users to share spaces and pages, the search scope includes email, a full user name, and user ID.

Procedure

1. On the Global security administrative console page, under **User account repository**, designate either **Federated repositories**, **Local Operating System**, **Standalone LDAP registry**, or **Standalone custom registry**.
2. Restart the server.
3. If you want to change the default user repository from the default **Federated repositories**, modify the `MashupAdminForOOBSpace` property in the `ConfigServices.properties` to designate the correct user ID (the UID property for your user repository) as the valid administrator ID.
   a. Copy the modified file into an empty folder on your system. The `ConfigServices.properties` file is located at *profile_root*`\BusinessSpace\`*node_name*`\`*server_name*`\mm.runtime.prof\config\ConfigService.properties` for a stand-alone server or *deployment_manager_profile_root*`\BusinessSpace\`*cluster_name*`\mm.runtime.prof\config\ConfigService.properties` for a cluster.
   b. Run the **updatePropertyConfig** command using the wsadmin scripting client.
      - For a stand-alone server:

        The following example uses Jython:

        ```
        AdminTask.updatePropertyConfig('[-serverName server_name -nodeName node_name
        -propertyFileName "profile_root\BusinessSpace\node_name\server_name
        \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"]')
        AdminConfig.save()
        ```

        The following example uses Jacl:

        ```
        $AdminTask updatePropertyConfig {-serverName server_name -nodeName node_name
         -propertyFileName "profile_root\BusinessSpace\node_name\server_name
        \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"}
        $AdminConfig save
        ```

- For a cluster:

  The following example uses Jython:

  ```
  AdminTask.updatePropertyConfig('[-clusterName cluster_name -propertyFileName
   "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
  config\ConfigService.properties" -prefix "Mashups_"]')
  AdminConfig.save()
  ```

  The following example uses Jacl:

  ```
  $AdminTask updatePropertyConfig {-clusterName cluster_name -propertyFileName
   "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
  config\ConfigService.properties" -prefix "Mashups_"}
  $AdminConfig save
  ```

c. Log into Business Space and reassign the owners of the default spaces to the new administrator ID.

4. If you are using an LDAP repository with a unique LDAP field, such as `mail` (email address) for the login property instead of `uid` (user ID), modify the `userIdKey` property in the `ConfigServices.properties` file in order for searching to work in Business Space.

   a. Locate the `ConfigServices.properties` file at *profile_root*`\BusinessSpace\`*node_name*`\`*server_name*`\mm.runtime.prof\config\ConfigService.properties` for a stand-alone server or *deployment_manager_profile_root*`\BusinessSpace\`*cluster_name*`\mm.runtime.prof\config\ConfigService.properties` for a cluster.

   b. Change the `userIdKey` attribute from `uid` to match the login property for your LDAP user repository, for example, `mail`.

   c. Copy the modified file into an empty folder on your system.

   d. Run the **updatePropertyConfig** command using the wsadmin scripting client.

   - For a stand-alone server:

     The following example uses Jython:

     ```
     AdminTask.updatePropertyConfig('[-serverName server_name -nodeName node_name
     -propertyFileName "profile_root\BusinessSpace\node_name\server_name
     \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"]')
     AdminConfig.save()
     ```

     The following example uses Jacl:

     ```
     $AdminTask updatePropertyConfig {-serverName server_name -nodeName node_name
      -propertyFileName "profile_root\BusinessSpace\node_name\server_name
     \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"}
     $AdminConfig save
     ```

   - For a cluster:

     The following example uses Jython:

     ```
     AdminTask.updatePropertyConfig('[-clusterName cluster_name -propertyFileName
      "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
     config\ConfigService.properties" -prefix "Mashups_"]')
     AdminConfig.save()
     ```

     The following example uses Jacl:

     ```
     $AdminTask updatePropertyConfig {-clusterName cluster_name -propertyFileName
      "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
     config\ConfigService.properties" -prefix "Mashups_"}
     $AdminConfig save
     ```

5. If you want to restrict logging in to Business Space to a subset of users and groups, you can change the mapping of the Business Space Java EE security role.

a. Update the user/group mapping for two enterprise applications: **BSpaceEAR**_*node_server* and **mm.was**_*node_server* (for a stand-alone server environment) or **BSpaceEAR**_*cluster* and **mm.was**_*cluster* (for a network deployment environment).
b. Click **Applications** > **Application Types** > **WebSphere enterprise applications** and select the two applications.
c. In the right panel, under Detail Properties, select **Security role to user/group mapping**.
d. Remap the **businessspaceusers** and **Allauthenticated** roles from the two applications by first removing the special subject.
e. Click **Map Special Subjects** and select **None**.
f. Click **Map Users** or **Map Groups** and assign each role to your selected users or groups.

Changing the Java EE security role mapping does not affect the user/group search function in Business Space.

6. Restart the server.
7. Log in to Business Space and reassign the owners of the default spaces to the new administrator ID.

What to do next

- To set authorization to pages and spaces in Business Space, you can manage authorization when creating Business Space pages and spaces.
- To designate who can perform Business Space administrator actions in the Business Space environment, see "Assigning the Business Space superuser role" on page 950.

Note:

If you find the following errors in the `SystemOut.log` file, you might have extra attributes in your user registry that cannot be processed:

```
00000046 SystemErr R Caused by: com.ibm.websphere.wim.exception.WIMSystemException: CWWIM1013E
    The value of the property secretary is not valid for entity uid=xxx,c=us,ou=yyy,o=ibm.com.
00000046 SystemErr R at com.ibm.ws.wim.adapter.ldap.LdapAdapter.setPropertyValue
(LdapAdapter.java:3338)
```

Set the following attributes in the `ConfigServices.properties` file to bypass those attributes:

```
com.ibm.mashups.user.userProfile = LIMITED
com.ibm.mashups.user.groupProfile = LIMITED
```

The `ConfigServices.properties` file is located at *profile_root*`\BusinessSpace\`*node_name*`\`*server_name*`\mm.runtime.prof\config\ConfigService.properties` for a stand-alone server or *deployment_manager_profile_root*`\BusinessSpace\`*cluster_name*`\mm.runtime.prof\config\ConfigService.properties` for a cluster. After modifying the `ConfigServices.properties` file, run the **updatePropertyConfig** command using the wsadmin scripting client by following the instructions in step 4.d. above.

Note:

If you have Java EE security enabled in a cluster, consider tightening the entry in the server policy applied to the Business Space help location.

The Business Space help location policy is:

```
grant codeBase     "file:${was.install.root}/profiles/profile_name/temp/
node_name/-" {

    permission java.security.AllPermission;

};
```

Tighten the policy by changing it to:

```
grant codeBase     "file:${was.install.root}/profiles/profile_name/temp/
node_name/server_name/BSpaceHelpEAR_node_name_server_name/BSpaceHelp.war/-"
{

    permission java.security.AllPermission;

 };
```

Setting up SSO and SSL for Business Space

For remote environments where Business Space and your product server are in different cells, set up single-sign-on (SSO) and Secure Sockets Layer (SSL) configuration manually.

Before you begin

Before you complete this task, you must have completed the following tasks:

- Enable application security and administrative security. See "Enabling security for Business Space" on page 934.
- Check that your user ID is registered in the user registry for your product.

Tip: If you have separate cells configured, make sure that SSO considerations are taken into account (including that LTPA keys are in synch, shared user names/realm names are in synch, and certificates are imported as appropriate). In some cases, with IBM Business Process Manager, there might be multiple repositories in the realm, which might result in a realm-mismatch error. See Managing the realm in a federated repository configuration in the WebSphere Application Server documentation.

Procedure

1. If Business Space is remote from where your product is running, and if the node where Business Space is running and the node where your product is running are not in the same cell, you must complete manual steps to make sure that SSO is enabled. For example, if you are using more than one product, the servers are on different nodes, and you want them all to be able to work with the Business Space server, you must manually configure SSO. To enable SSO, complete the following steps:
   a. On the administrative console for each server, open the Global security page by clicking **Security** > **Global security**. Expand **Web and SIP security** and click **single sign-on (SSO)** to make sure that the **Enabled** check box is selected.
   b. Make sure that all the nodes use the same **User account repository** information (see step 3).
   c. Follow the steps in Import and export keys in the WebSphere Application Server information center.

2. If you are using HTTPS in the endpoints file, the endpoint location is on a different node than Business Space, and the SSL certificate is a self-signed SSL certificate, you must import it.

   Make sure that the signers are configured in the appropriate truststores for the Business Space server and the product server. See Secure communications using Secure Sockets Layer (SSL) in the WebSphere Application Server information center.

   For more information about SSO and SSL, see the WebSphere Application Server information center.

Designating HTTP or HTTPS settings for Business Space

Business Space is configured to be accessed by HTTPS by default. If you want to change to HTTP, or you want to change back to HTTPS from a previous setting, use the `configBSpaceTransport.py` script.

About this task

The `configBSpaceTransport.jy` script is located in the *install_root*\`BusinessSpace\scripts` directory. Use the `allowhttp` value to change the `web.xml` to allow HTTP and HTTPS connections. Use the `httpsonly` value to change `web.xml` to allow only HTTPS and to redirect HTTP to HTTPS. Run the script on your stand-alone server or on the deployment manager for a clustered environment.

Procedure

1. Locate the script *install_root*\`BusinessSpace\scripts\configBSpaceTransport.py` for designating HTTP or HTTPS settings.
2. Open a command prompt, and change directories to the following directory: *profile_root*\`bin`, where *profile_root* represents the directory for the profile where Business Space is installed.
3. Run the `configBSpaceTransport.py` with one the following options.
   a. To allow HTTP connections to Business Space, type the following command: `wsadmin -lang jython -user` *user_name* `-password` *password* `-f configBSpaceTransport.py -allowhttp` (allows both HTTP and HTTPS).
   b. To allow only HTTPS connections to Business Space, run: `wsadmin -lang jython -user` *user_name* `-password` *password* `-f configBSpaceTransport.py -httpsonly` (switches back to the default configuration, which always redirects HTTP to HTTPS).

   By default, the command applies to the current server name and node name, or on the current cluster, where you are running the command. If you want to designate a different location, use the optional `-serverName` and `-nodeName` parameters or the `-clusterName` parameter.

   Note: When the path contains a space, for example, if *install_root* is `My install dir`, you must enclose the path names in quotation marks.

Setting up security for system REST services

To set up security for the data in the widgets based on users and groups, you must modify the users that are mapped to the REST services gateway application.

Before you begin

Before you complete this task, you must have completed the following tasks:

- Enable application security and administrative security. See "Enabling security for Business Space" on page 934.
- Check that your user ID is registered in the user registry for your product.

About this task

How you map users to a REST service provider application affects all the services for the provider.

To see the affected services, select **Services** > **REST services** > **REST service providers**, and select the matching provider application in the list of providers.

Procedure

1. On the administrative console, select one of the following options:
   - For a server environment, select **Applications** > **Application types** > **WebSphere enterprise applications** > **REST Services Gateway**
   - In addition, for a network deployment environment, select **Applications** > **Application types** > **WebSphere enterprise applications** > **REST Services Gateway Dmgr**
2. In the right panel, under Detail Properties, select **Security role to user/group mapping**.
3. To control access to the data in all the REST services widgets, add users and groups to the **RestServicesUser** role.

Business Space widget security considerations

Depending on the widgets you use in Business Space with your business process management product, you might assign either administrative user group roles to control access to data in a widget, or you might assign an additional layer of role-based access for your widget.

Administrative group roles and widgets

You control access to data in widgets with administrative group roles and the users who are assigned to the administrative group roles. To see who is assigned to these roles, open the administrative console, select **Users and groups** > **Administrative group roles**, and select a group. The Roles list is displayed.

Business Rules and Business Variables are two examples of widgets that might require changes to the administrative group roles.

For the System Health widget, the following administrative roles all have monitoring permissions, allow access to the administrative console and, therefore, allow users assigned to those roles to access data in the System Health widget:

- **Monitor**
- **Configurator**
- **Operator**
- **Administrator**
- **Adminsecuritymanager**
- **Deployer**
- **iscadmins**

Users who are mapped to those administrative group roles have access to the data in the System Health widget. Users who are not mapped to those roles cannot access the data in the System Health widget.

Widget role-based access

Configuring Tivoli Access Manager WebSEAL to work with Business Space

If you have Tivoli Access Manager WebSEAL and you want to use it with Business Space, you must complete several additional configuration steps.

Before you begin

Before you complete this task, you must have completed the following tasks:

- Enable application security and administrative security. See "Enabling security for Business Space" on page 934.
- Check that your user ID is registered in the user registry for your product.

About this task

If you want to use Tivoli Access Manager WebSEAL with Business Space, you must configure Tivoli Access Manager security to enable an external Java Authorization Contract for Containers (JACC) provider, configure WebSEAL to work with Tivoli Access Manager, configure WebSEAL to work with your product application server, and configure host junctions for your environment.

Procedure

1. Configure Tivoli Access Manager to enable an external JACC provider.
   a. Complete one of the following steps, depending on whether you want to use the administrative console or the wsadmin commands.
      - If you want to use the administrative console to configure Tivoli Access Manager to enable JACC, complete the following steps:
        1) Enable Global Security.
           a) Select **Security** > **Global Security**.
           b) Enable **Administrative security**, **Application security**, and **Java 2 security** with the LDAP server with which Tivoli Access Manager is configured.
           c) Select **Global Security** > **LDAP**, enter the following information, and then click **OK**.

Name	Description
`Server user Id`	Enter the same user ID that you entered for the administrator DN on Tivoli Access Manager settings. Example: `user1`
`Server user password`	`puser1`
`Host`	LDAP configured with Tivoli Access Manager
`Port`	Example: `389`
`Base DN`	Example: `o=ibm, c=us`
`Bind DN`	Example: `cn=SecurityMaster,secAuthority=Default`

Name	Description
`Bind pwd`	password for SecurityMaster user

d) Save the configuration, and restart the server.

2) Enable external authorization with Tivoli Access Manager and JACC.

a) Select **Security** > **Global Security** > **External authorization providers**.

b) In the **Authorization provider** list, select **External JACC provider**, and then click **Configure**. The default properties for Tivoli Access Manager are correct. For default values, do not change.

c) Under **Additional Properties**, select **Tivoli Access Manager properties**. Select **Enable embedded Tivoli Access Manager**, enter the following information, and then click **OK**.

Name	Value
Client listening port set	The default setting is 8900 - 8999. Change it only if you want to use different ports.
Policy server (name:port)	Specify your *policyserver*:*port*. Example: `windomain3.rtp.raleigh.ibm.com:7135`
Authorization servers and priority (name:port:priority)	Specify your *authorizationserver*:*port*:*priority*. Example: `windomain3.rtp.raleigh.ibm.com:7136:1`
Administrator user name	Leave the user name as **sec_master (default)**, unless you use a different admin name on the Tivoli Access Manager server.
Administrator user password	`domino123`
User registry distinguished name suffix	Type the name that you want to use for your application server. Example: `o=ibm,c=us`
Security domain	Leave the Security domain set to **Default**. Change this setting if you are not using the default domain on the Tivoli Access Manager server. Change this setting if you have multiple domains created on the Tivoli Access Manager server and you want to connect or use a domain other than **Default**.
Administrator user distinguished name	Type the fully qualified name of the user. Example: `cn=user1,o=ibm,c=us` **Note:** This user is the same as the **Server user ID** configured in the LDAP user registry panel.

The server contacts the Tivoli Access Manager server and creates several properties files under the application server. This process might take a few minutes. If an error occurs, look in `system Out` and correct the problem.

- If you want to use the wsadmin utility to configure Tivoli Access Manager to enable JACC, complete the following steps. Perform the following procedure once on the deployment manager server. The configuration parameters are forwarded to managed servers, including node agents, when a synchronization is performed. The managed servers require their own restart for the configuration changes to take effect.

1) Verify that all the managed servers, including node agents, are started.
2) Start the server.
3) Start the command-line utility by running the **wsadmin** command from the *install_root*/bin directory.
4) At the wsadmin prompt, run the **configureTAM** command, including the appropriate information from the following table:

   Jacl example:

   ```
   $AdminTask configureTAM -interactive
   ```

   Jython example:

   ```
   AdminTask.configureTAM('-interactive')
   ```

   Then type the following information:

Name	Value
node name for your product server	Specify a single node or enter an asterisk (*) to choose all nodes.
Tivoli Access Manager Policy Server	Type the name of the Tivoli Access Manager policy server and the connection port. Use the format, *policy_server*:*port*. The policy server communication port is set at the time of Tivoli Access Manager configuration. The default port is 7135.
Tivoli Access Manager Authorization Server	Type the name of the Tivoli Access Manager authorization server. Use the format *auth_server*:*port*:*priority*. The authorization server communication port is set at the time of Tivoli Access Manager configuration. The default port is 7136. You can specify more than one authorization server by separating the entries with commas. Having more than one authorization server configured is useful for failover and performance. The priority value is the order of authorization server use. For example: `auth_server1:7136:1,auth_server2:7137:2`. A priority of 1 is still required when configuring against a single authorization server.
administrator distinguished name for your product server	Type the full distinguished name of the security administrator ID for your product server. For example: `cn=wasadmin,o=organization,c=country`. For more information, see the related link.
Tivoli Access Manager user registry distinguished name suffix	For example: o=organization, c=country
Tivoli Access Manager administrator user name	Type the Tivoli Access Manager administration user ID, as created at the time of Tivoli Access Manager configuration. This ID is typically sec_master.
Tivoli Access Manager administrator user password	Type the password for the Tivoli Access Manager administrator.

Name	Value
Tivoli Access Manager security domain	Type the name of the Tivoli Access Manager security domain that is used to store users and groups. If a security domain is not already established at the time of Tivoli Access Manager configuration, click **Return** to accept the default.
Embedded Tivoli Access Manager listening port set	The product server listens on a TCP/IP port for authorization database updates from the policy server. Because more than one process can run on a particular node and machine, a list of ports is required for the processes. Specify the ports that are used as listening ports by Tivoli Access Manager clients, separated by a comma. If you specify a range of ports, separate the lower and higher values by a colon. For example, 7999, 9990:9999.
Defer	Set to `yes`, this option defers the configuration of the management server until the next restart. Set to `no`, configuration of the management server occurs immediately. Managed servers are configured on their next restart.

5) After you enter all the required information, select **F** to save the configuration properties or **C** to cancel from the configuration process and discard the entered information.

   Example with SVTM TAM60 server:

```
wsadmin>$AdminTask configureTAM -interactive
Configure embedded Tivoli Access Manager

This command configures embedded Tivoli Access Manager on the WebSphere
Application Server node or nodes specified.

WebSphere Application Server Node Name (nodeName): *
*Tivoli Access Manager Policy Server (policySvr):
 windomain3.rtp.raleigh.ibm.com:7135
*Tivoli Access Manager Authorization Servers (authSvrs):
 windomain3.rtp.raleigh.ibm.com:7136:1
*WebSphere Application Server administrator's distinguished name (wasAdminDN):
 cn=was61admin,o=ibm,c=us
*Tivoli Access Manager user registry distinguished name suffix (dnSuffix):
o=ibm,c=us
Tivoli Access Manager administrator's user name (adminUid):
 [sec_master]
*Tivoli Access Manager administrator's user password (adminPasswd):
 domino123
Tivoli Access Manager security domain (secDomain): [Default]
Embedded Tivoli Access Manager listening port set (portSet): [9900:9999]
Defer (defer): [no]

Configure embedded Tivoli Access Manager

F (Finish)
C (Cancel)

Select [F, C]: [F] F
WASX7278I: Generated command line: $AdminTask configureTAM {-policySvr
 windomain3.rtp.raleigh.ibm.com:7135 -authSvrs
 windomain3.rtp.raleigh.ibm.com:7136:1 -wasAdminDN cn=wa
Embedded Tivoli Access Manager configuration action parameters saved successfully.
 Restart all WebSphere Application Server instances running on the target node or
 nodes to
wsadmin>
```

6) In the administrative console, select **Security** > **Global Security** > **External authorization providers**. Then select **External authorization using a JACC provider**, and click **OK**.
7) Go to the main security screen and click **OK**. Save and synchronize your changes.
8) Restart all processes in your cell.

b. If you installed applications before you enabled Tivoli Access Manager (for example, you enabled LDAP security and installed some secured applications and mapped users and groups to security roles), propagate the security roles mapping information from the deployment descriptors to the Tivoli Access Manager policy server. Perform one of the following steps, depending on whether you want to use the administrative console, or the wsadmin commands.
   - If you want to use the **propagatePolicyToJACCProvider** wsadmin command, see Propagating security policy of installed applications to a JACC provider using wsadmin scripting.
   - If you want to use the administrative console, see Propagating security policies and roles for previously deployed applications.

2. Configure WebSEAL to work with Tivoli Access Manager.
   a. Ensure that WebSEAL is installed and configured properly.
   b. To create a trusted user account in Tivoli Access Manager, which can be used for configuring TAI, issue the following commands:

```
pdadmin -a sec_master -p domino123
pdadmin sec_master> user create -gsouser -no-password-policy taiuser
"cn=taiuser,ou=websphere,o=ibm,c=us" taiuser taiuser ptaiuser
pdadmin sec_master> user modify taiuser password-valid yes
pdadmin sec_master> user modify taiuser account-valid yes
```

   c. Create the junction between WebSEAL and your product application server using the **-c iv_creds** option for TAI++ and **-c iv_user** for TAI. Enter either of the following commands as one line, using the variables that are appropriate for your environment:

      For TAI++

```
server task webseald-server create -t tcp -b supply -c iv_creds
-h host_name -p websphere_app_port_number /junction_name
```

      Tip: The *junction_name* must begin with /.

   d. In the WebSEAL configuration file *webseal_install_directory*/etc/webseald-default.conf, set the following parameter:

```
basicauth-dummy-passwd=webseal_userid_passwd
```

      For example, if you set the taiuser/ptaiuser in Tivoli Access Manager, set the following parameter:`basicauth-dummy-passwd = ptaiuser`

      If you are using a form-based authentication, set the following parameters:

```
forms-auth=both
ba-auth=none
```

3. If needed, configure WebSEAL to work with your product application server by enabling the TAI++ interceptor on the server.
   a. In the administrative console, select **Global security** > **Authentication mechanisms and expiration**.
   b. Expand **Web and SIP security**, and then select **Trust Association**. Select the check box and click **Apply**.

c. Select **Interceptors** > **TAMTrustAssociationInterceptorPlus** > **custom properties**, and add the following properties:

Name	Value
`com.ibm.websphere.security.webseal.configURL`	`${WAS_INSTALL_ROOT}/java/jre/PdPerm.properties`
`com.ibm.websphere.security.webseal.id`	`iv-creds`
`com.ibm.websphere.security.webseal.loginId`	`taiuser` (if the user `taiuser/ptaiuser` was created in the Tivoli Access Manager)

d. Restart the cell.

e. To access the client, go to https://*webseal_server_name*:*webseal_port*/*junction name*/*web_uri_for_client*.

4. Configure the host junctions for your environment, so that the Business Space widgets appear. Complete one of the following steps, depending on whether you are using virtual host junctions or transparent host junctions.

   - If you are using virtual host junctions, create a virtual host junction. A virtual host junction eliminates the need to create separate junctions.

     a. Make sure that a virtual host has been configured. Virtual host junctions match a host and port number and forward addresses to the target host. No URL filtering occurs, and all requests that match are forwarded to the target host.

     b. Make sure that the following applications are available to the same virtual host. You may have some or all of the applications, based on which products you are using with Business Space.

        – BPMAdministrationWidgets_*nodename_servername* (for WebSphere Enterprise Service Bus and IBM Business Process Manager)
        – BusinessSpaceHelpEAR_*nodename_servername* (for all products)
        – BSpaceEAR_*nodename_servername* (for all products)
        – BSpaceForms_*nodename_servername* (for all products)
        – HumanTaskManagementWidgets_*nodename_servername* (for IBM Business Process Manager and IBM Business Monitor)
        – PageBuilder2_*nodename_servername* (for all products)
        – REST Services Gateway (for all products)
        – REST Services Gateway Dmgr (for WebSphere Enterprise Service Bus and IBM Business Process Manager)
        – mm.was_*nodename_servername* (for all products)
        – WBMDashboardWeb_*nodename_servername* (for IBM Business Monitor)
        – wesbWidgets_*nodename_servername* (for WebSphere Enterprise Service Bus)

        Note: This list of applications covers only the applications required by Business Space. You might need to add other applications to the list for non-Business Space scenarios using Tivoli Access Manager WebSEAL.

     c. Run the following command using pdadmin: `server task` *`webseal server`* `virtualhost create -t` *`transport`* `-h` *`target_host`* `[-p` *`port`*`] [-v` *`virtual_host_name`*`]` *`virtual_host_label`*

        Use the following information:

        – *webseal server* is the name of the WebSEAL server where you will create the virtual host entry.

– *transport* is the type of transport. Valid entries are `tcp`, `ssl`, `tcpproxy`, and `sslproxy`.
– *target_host* is the host of the required application.
– *virtual_host_name* is used to match HTTP requests to a virtual host junction. If no value is entered, it is made up of the target host and port by default. For example, if you set the *virtual_host_name* to `myvirthost.ibm.com:80`, WebSEAL matches the URLs containing `myvirthost.ibm.com:80` and routes it to the host provided in the pdadmin command.
– *virtual_host_label* is the label used to identify the entry in WebSEAL. It must be unique.

For Business Space to run as expected, both `ssl` and `tcp` entries must be created for the type of transport. When you need both Secure Sockets Layer (SSL) and Transmission Control Protocol (TCP) to be supported in the same virtual host junction, you must us the -g *vhost_label* option, where *vhost_label* is the original virtual host label to share configuration. This option finds a previously created virtual host junction (one created earlier, where the *virtual_host_label* matches the label provided in the -g option), and will share that configuration. The second entry still needs its own *virtual_host_label*, but it can share the target host, port, and other values. If you do not provide this -g option, a second virtual host cannot be created because WebSEAL will see the target host and port as being identical to a previously create junction (which is not allowed).

- If you are using transparent host junctions, create a series of transparent path junctions for the widgets for each product.
  a. Review each context root you have defined. See Mapping Business Space URLs for a reverse proxy server).
  b. For each context root defined, run the following command using padmin: `server task webseal server create -t transport type (ssl) or (tcp) -x -h hostname path`.

     For example, type: `server task webseald-default create -t tcp -x -h monServer.ibm.com /BusinessSpace`.
  c. Update the following two properties in ConfigService.properties of the Business Space server:

     ```
     reverseProxyHost = WebSEAL host
     reverseProxyPort = WebSEAL port, for example: 80
     ```
  d. Run the **updatePropertyConfig** command using the wsadmin scripting client.
     – For a stand-alone server:

       The following example uses Jython:

       ```
       AdminTask.updatePropertyConfig('[-serverName server_name
       -nodeName node_name -propertyFileName "profile_root\
       BusinessSpace\node_name\server_name\mm.runtime.prof\config\
       ConfigService.properties" -prefix "Mashups_"]')
       AdminConfig.save()
       ```

       The following example uses Jacl:

       ```
       $AdminTask updatePropertyConfig {-serverName server_name
       -nodeName node_name -propertyFileName "profile_root\
       BusinessSpace\node_name\server_name\mm.runtime.prof\config\
       ConfigService.properties" -prefix "Mashups_"}
       $AdminConfig save
       ```

– For a cluster:

The following example uses Jython:

```
AdminTask.updatePropertyConfig('[-clusterName cluster_name
-propertyFileName "deployment_manager_profile_root\
BusinessSpace\cluster_name\mm.runtime.prof\config\
ConfigService.properties" -prefix "Mashups_"]')
AdminConfig.save()
```

The following example uses Jacl:

```
$AdminTask updatePropertyConfig {-clusterName cluster_name
-propertyFileName "deployment_manager_profile_root\
BusinessSpace\cluster_name\mm.runtime.prof\config\
ConfigService.properties" -prefix "Mashups_"}
$AdminConfig save
```

5. Complete additional configuration steps to resolve issues with browser cookies and virtual hosts.
   a. To resolve renaming of the Business Space cookie, add the following content to the WebSEAL configuration file:

      ```
      [preserve-cookie-names]
      name = com.ibm.bspace.UserName
      name = com.ibm.wbimonitor.UserName
      ```

   b. Optional: If you are using non-default virtual hosts with a context root, you might encounter issues with Business Space pages. You might need to stop the junction from rewriting the JavaScript on the Business Space pages by adding the -j junction to the context root. Run the following command:
      `server task default-webseald create -f -h hostname -p portnumber -t tcp -b supply -c iv-user,iv-creds,iv-groups -x -s -j -J trailer/context_root`

Assigning the Business Space superuser role

In Business Space, you can assign users to be superusers (or Business Space administrators). A superuser can view, edit, and delete all spaces and pages, can manage and create templates, and can change ownership of a space by changing the owner ID.

Before you begin

Before you complete this task, you must have completed the following tasks:

- Enable application security and administrative security. See "Enabling security for Business Space" on page 934.
- Check that your user ID is registered in the user registry for your product.

About this task

Assign the Business Space superuser role by using the following application server security role: **Admin**. Using this method gives you flexibility in assigning the role to any number of your organization's existing groups and users. It doesn't require the creation of an administrators group in the user registry for the sole purpose of acting as the focal point for the Business Space superuser.

If you already have the Business Space superuser assigned from an earlier version than V7.5, you can modify the superuser by user group instead. See Assigning the Business Space superuser by user group.

Procedure

- If you are setting up the Business Space administrators with the superuser role for the first time, complete the following steps.
  1. Log in to the administrative console for your product.
  2. Click **Applications** > **Application Types** > **WebSphere enterprise applications** and select one of the following applications:
     - **mm.was_*node_server*** (for a stand-alone server environment)
     - **mm.was_*cluster*** (for a network deployment environment)
  3. Click **Security role to user/group mappings**.
  4. Select the row for the **Admin** role, and click the **Map Users** button or the **Map Groups** button to map either users or groups to the Admin role.
  5. Click **Save**.
  6. Restart the server.
- If you previously assigned superusers based on user groups, and you want to switch to this simpler way to manage superusers by role, complete the following steps.
  1. Open the configuration file.
     - For a stand-alone server: *profile_root*\BusinessSpace*node_name**server_name*\mm.runtime.prof\config\ConfigService.properties
     - For a cluster: *deployment_manager_profile_root*\BusinessSpace*cluster_name*\mm.runtime.prof\config\ConfigService.properties
  2. Change the following property values in the configuration file.

     ```
     com.ibm.mashups.adminGroupName = {com.ibm.mashups.J2EERole.Admin}
     com.ibm.mashups.widget.attributes.configure.groups=
     ```
  3. Run the **updatePropertyConfig** command in the wsadmin environment of the profile.
     - For a stand-alone server:

       The following example uses Jython:

       ```
       AdminTask.updatePropertyConfig('[-serverName server_name -nodeName node_name
       -propertyFileName "profile_root\BusinessSpace\node_name\server_name
       \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"]')
       AdminConfig.save()
       ```

       The following example uses Jacl:

       ```
       $AdminTask updatePropertyConfig {-serverName server_name -nodeName node_name
        -propertyFileName "profile_root\BusinessSpace\node_name\server_name
       \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"}
       $AdminConfig save
       ```
     - For a cluster:

       The following example uses Jython:

       ```
       AdminTask.updatePropertyConfig('[-clusterName cluster_name -propertyFileName
        "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
       config\ConfigService.properties" -prefix "Mashups_"]')
       AdminConfig.save()
       ```

       The following example uses Jacl:

       ```
       $AdminTask updatePropertyConfig {-clusterName cluster_name -propertyFileName
        "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
       config\ConfigService.properties" -prefix "Mashups_"}
       $AdminConfig save
       ```
  4. Restart the server.
  5. Use the step above to assign users to the Business Space superuser roles.

Assigning the Business Space superuser by user group

You can assign Business Space users to be superusers (or Business Space administrators) based on user groups. This is useful if you worked in version earlier than V7.5, and the superuser role is already defined by user group.

Before you begin

Before you complete this task, you must have completed the following tasks:

- Enable application security and administrative security. See "Enabling security for Business Space" on page 934.
- Check that your user ID is registered in the user registry for your product.

Tip: If you previously used user groups to assign the Business Space superuser role, you can switch to the simpler way to assign Business Space superusers by role. See "Assigning the Business Space superuser role" on page 950.

A superuser can view, edit, and delete all spaces and pages, can manage and create templates, and can change ownership of a space by changing the owner ID.

If administrative security is enabled when you configure Business Space, consider the following information about groups and superusers:

- Users belonging to the special user group, **administrators**, have a superuser role by default. As a result, the superuser role assignment is handled by user group membership.
- In a single-server environment, the Business Space server creates the **administrators** user group in the default user registry. The administrator ID provided during configuration is automatically added as member of this group.
- In a network deployment environment, the **administrators** user group is not created automatically. Use the `createSuperUser.py` script to create the user group and add members to that group in the default user registry.
- If another user registry (for example, LDAP) is used instead of the default user registry, or if the default user registry is used but you do not want to use the **administrators** user group, you must identify the user group that you are using for the Business Space superusers. Make sure that the value you provide can be understood by the user registry. For example, for LDAP, you might provide a name like `cn=administrators,dc=company,dc=com`. For more information about identifying this user group, see the instructions for changing the administrators group in the What to do next section.
- For Business Space in WebSphere Portal, the default group **wpsadmins** is also used for the superuser role. Members of this group are granted the superuser role for Business Space.

  Note: Security must be enabled if you want to use Business Space in WebSphere Portal.

If administrative security is not enabled when you configure Business Space, only the special user ID **BPMAdministrator** has the Business Space superuser role.

If you have a network deployment environment, you must run the `createSuperUser.py` script to assign the superuser role: to create the user group and add members. Before you run the script, complete the following steps:

- Make sure the default **administrators** group name is not changed.
- Use the default repository for the user registry.

- Start the server or the deployment manager for your Business Space environment for the profile where is Business Space installed.

Procedure

1. Locate the script *install_root*\BusinessSpace\scripts\createSuperUser.py for assigning the superuser role to a user.
2. Open a command prompt, and change directories to the following directory: *profile_root*\bin, where *profile_root* represents the directory for the profile where Business Space is installed.
3. Type the following command: wsadmin -lang jython -f *install_root*\BusinessSpace\scripts\createSuperUser.py *user_short_name password* where *user_short_name* is the unique identifier for a user in Virtual Member Manager (VMM), and *password* is the VMM password for that user. If that user exists in VMM, the user is added to the administrator group.

   Note: When the path contains a space, for example, if *install_root* is My install dir, you must enclose the path names in quotation marks. For example, type the following command: wsadmin -lang jython -f "\My install dir\BusinessSpace\scripts\createSuperUser.py" *user_short_name_in_VMM*.

What to do next

To open Business Space, use the following URL: http://*host*:*port*/BusinessSpace, where *host* is the name of the host where your server is running and *port* is the port number for your server.

You can change the default special user group named **adminstrators**. Perform the following steps to check the current group name or change it to other name.

Inspect the value for the metric com.ibm.mashups.adminGroupName in the configuration file:

- *profile_root*\BusinessSpace*node_name**server_name*\mm.runtime.prof\config\ConfigService.properties on a stand-alone server, or
- *deployment_manager_profile_root*\BusinessSpace*cluster_name*\mm.runtime.prof\config\ConfigService.properties on a cluster.

If you want to change an administrative group, perform the following steps on a stand-alone server:

1. Make sure that the group exists in the user repository.
2. Modify the metric com.ibm.mashups.adminGroupName in the configuration file *profile_root*\BusinessSpace*node_name**server_name*\mm.runtime.prof\config\ConfigService.properties.
3. Run the command updatePropertyConfig in the wsadmin environment of the profile:$AdminTask updatePropertyConfig {-serverName *server_name* -nodeName *node_name* -propertyFileName "*profile_root*\BusinessSpace*node_name**server_name*\mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"} and run $AdminConfig save.
4. Restart the server.

If you want to change an administrative group, perform the following steps on a cluster:

1. Make sure that the group exists in the user repository.

2. Modify the metric `com.ibm.mashups.adminGroupName` in the configuration file *deployment_manager_profile_root*`\BusinessSpace\`*cluster_name*`\mm.runtime.prof\config\ConfigService.properties`.
3. Run the command updatePropertyConfig in the wsadmin environment of the deployment environment profile:`$AdminTask updatePropertyConfig {-clusterName` *cluster_name* `-propertyFileName "`*deployment_manager_profile_root*`\BusinessSpace\`*cluster_name*`\mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"}` and run `$AdminConfig save`.
4. Restart the deployment manager.

If you want to change the superuser when security is not enabled, perform the following steps on a stand-alone server:

1. Modify the metric `noSecurityAdminInternalUserOnly` in the configuration file *profile_root*`\BusinessSpace\`*node_name*`\`*server_name*`\mm.runtime.prof\config\ConfigService.properties`.
2. Run the command updatePropertyConfig in the wsadmin environment of the profile:`$AdminTask updatePropertyConfig {-serverName` *server_name* `-nodeName` *node_name* `-propertyFileName "`*profile_root*`\BusinessSpace\`*node_name*`\`*server_name*`\mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"}` and run `$AdminConfig save`.
3. Restart the server.

If you want to change the superuser when security is not enabled, perform the following steps on a cluster:

1. Modify the metric `noSecurityAdminInternalUserOnly` in the configuration file *deployment_manager_profile_root*`\BusinessSpace\`*cluster_name*`\mm.runtime.prof\config\ConfigService.properties`.
2. Run the command updatePropertyConfig in the wsadmin environment of the deployment environment profile:`$AdminTask updatePropertyConfig {-clusterName` *cluster_name* `-propertyFileName "`*deployment_manager_profile_root*`\BusinessSpace\`*cluster_name*`\mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"}` and run `$AdminConfig save`.
3. Restart the deployment manager.

Preventing users from creating business spaces

You can customize Business Space so that only users logging in with a superuser role can create business spaces.

About this task

By default, all users can create business spaces. However, you can lock down Business Space so that only people who log in using a superuser ID can create or import business spaces. These superusers (or Business Space administrators) can create a business space and transfer ownership to other users. The users who are assigned ownership of spaces can then administer the spaces as if they had created them. For example, they can set who can view and edit the space and its properties and they can add pages.

Note: The ability to prevent users from creating business spaces is not available for Business Space running on WebSphere Portal Server.

To limit creating business spaces to superusers only, complete the following steps.

Procedure

1. Change the `com.ibm.mashups.lockeddown` setting to `true` in the configuration file:
   - For a stand-alone server: *profile_root*\BusinessSpace*node_name**server_name*\mm.runtime.prof\config\ConfigService.properties
   - For a cluster: *deployment_manager_profile_root*\BusinessSpace*cluster_name*\mm.runtime.prof\config\ConfigService.properties

   The default value of `false` means that all users can create business spaces. When the value is `true`, only superusers can create business spaces.
2. Run the **updatePropertyConfig** command in the `wsadmin` environment of the profile:
   - For a stand-alone server:

     The following example uses Jython:

     ```
     AdminTask.updatePropertyConfig('[-serverName server_name -nodeName node_name
     -propertyFileName "profile_root\BusinessSpace\node_name\server_name
     \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"]')
     AdminConfig.save()
     ```

     The following example uses Jacl:

     ```
     $AdminTask updatePropertyConfig {-serverName server_name -nodeName node_name
      -propertyFileName "profile_root\BusinessSpace\node_name\server_name
     \mm.runtime.prof\config\ConfigService.properties" -prefix "Mashups_"}
     $AdminConfig save
     ```
   - For a cluster:

     The following example uses Jython:

     ```
     AdminTask.updatePropertyConfig('[-clusterName cluster_name -propertyFileName
      "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
     config\ConfigService.properties" -prefix "Mashups_"]')
     AdminConfig.save()
     ```

     The following example uses Jacl:

     ```
     $AdminTask updatePropertyConfig {-clusterName cluster_name -propertyFileName
      "deployment_manager_profile_root\BusinessSpace\cluster_name\mm.runtime.prof\
     config\ConfigService.properties" -prefix "Mashups_"}
     $AdminConfig save
     ```

Results

The next time that users log into Business Space, they will not be able to create a business space unless they log in using a superuser ID.

Enabling searches for user registries without wildcards

If your user registry is set up to not use wildcards, you must complete additional configuration steps so that searches work properly in Business Space and for widgets that search the user registry.

Before you begin

Before you complete this task, you must have completed the following tasks:

- Enable application security and administrative security. See "Enabling security for Business Space" on page 934.
- Check that your user ID is registered in the user registry for your product.

About this task

By default, when a Business Space user searches for users or groups by typing one or more characters, Business Space automatically adds wildcard characters. For example, if the user registry is an LDAP server and the user types `smit`, Business Space converts this into a `*smit*` query so that the return includes names like `Smith`, `Smithers`, and `Psmith`. However, if you do not want the automatic wildcards because, for example, your user registry does not permit them, you can disable this functionality.

To turn off the automatic wildcard searches for your environment, complete the following steps.

Procedure

- For a stand-alone server, complete the following steps:
  1. Update the *profile_root*\BusinessSpace*node_name**server_name*\mm.runtime.prof\config\ConfigService.properties configuration file with `com.ibm.mashups.user.stripWildcards=true`.
  2. Run the **updatePropertyConfig** command in the wsadmin environment of the profile:

     The following example uses Jython:

     ```
     AdminTask.updatePropertyConfig('[-serverName server_name -nodeName
     node_name -propertyFileName "profile_root\BusinessSpace\node_name\
     server_name\mm.runtime.prof\config\ConfigService.properties" -prefix
     "Mashups_"]')
     AdminConfig.save()
     ```

     The following example uses Jacl:

     ```
     $AdminTask updatePropertyConfig {-serverName server_name -nodeName
     node_name -propertyFileName "profile_root\BusinessSpace\node_name\
     server_name\mm.runtime.prof\config\ConfigService.properties" -prefix
     "Mashups_"}
     $AdminConfig save
     ```
  3. Restart the server.
- For a cluster, complete the following steps:
  1. Update the *deployment_manager_profile_root*\BusinessSpace*cluster_name*\mm.runtime.prof\config\ConfigService.properties configuration file with `com.ibm.mashups.user.stripWildcards=true`.
  2. From the deployment manager, run the **updatePropertyConfig** command in the wsadmin environment of the profile:

     The following example uses Jython:

     ```
     AdminTask.updatePropertyConfig('[-clusterName cluster_name
     -propertyFileName "deployment_manager_profile_root\BusinessSpace\
     cluster_name\mm.runtime.prof\config\ConfigService.properties" -prefix
     "Mashups_"]')
     AdminConfig.save()
     ```

     The following example uses Jacl:

     ```
     $AdminTask updatePropertyConfig {-clusterName cluster_name
     -propertyFileName "deployment_manager_profile_root\BusinessSpace\
     cluster_name\mm.runtime.prof\config\ConfigService.properties" -prefix
     "Mashups_"}
     $AdminConfig save
     ```

3. Restart the deployment manager.

Chapter 9. Service component monitoring overview

A conceptual overview of the reasons you monitor service components on the process server; which event points within the service components you select to monitor; and, how to configure monitoring on your system.

WebSphere ESB provides capabilities for monitoring service components to aid in system administration functions, such as performance tuning and problem determination. It goes beyond these traditional functions by also providing the capability for persons who are not necessarily information technology specialists to continually monitor the processing of the service components within the applications deployed on your system. By overseeing the overall processing flow of the interconnected components, you can ensure that your system is producing what you expect it to produce.

WebSphere ESB operates on top of an installation of WebSphere Application Server, and, consequently, uses much of the functionality of the application server infrastructure for monitoring system performance and troubleshooting. It also includes some extra functionality that is designed for monitoring process server service components. This section focuses on how you monitor server-specific service components. It is intended to supplement the monitoring and troubleshooting topics found in the WebSphere Application Server Information Center; therefore; refer to that documentation for details of the other monitoring capabilities in the combined product.

Why use monitoring?

You monitor service components within WebSphere ESB to assess performance, to troubleshoot problems, and to evaluate the overall processing progress of service components that make up the applications deployed on your system.

Service components are the integral functions incorporated into WebSphere ESB, with which you can create and deploy applications on your system that mirror the processes employed in your enterprise. Effectively monitoring those service components is, therefore, essential to managing the tasks that the server is intended to accomplish. There are three main reasons you need to monitor service components on the server:

Problem determination
: You can diagnose particular errors by using the logging and tracing facilities provided by WebSphere Application Server, which underlies WebSphere ESB. For example, if a particular application is not producing the expected results, you can set up a logger to monitor the processing of the service components that make up that application. You can have the log output published to a file, which you can then examine to pinpoint the cause of the problem. Troubleshooting is a task that is of importance to system administrators and others concerned with the maintenance of system hardware and software.

Performance tuning
: You can monitor certain performance statistics that most process server-specific service components produce. Use this information to maintain and tune your system health, and ensure that your applications are tuned optimally and efficiently. You can also spot situations where one

© Copyright IBM Corp. 2007, 2011

or more of your services are performing at a poor level, which may indicate that other problems are present in your system. Like problem determination, performance tuning is a task typically performed by information technology specialists.

Assessing the processing of service components
Problem determination and performance tuning are tasks you perform on a short-term basis, to solve a particular issue or problem. You can also set up the process server to continually monitor the service components incorporated into the applications deployed on your system. This type of service component monitoring is of importance to those who are responsible for designing, implementing, and ensuring that the processes achieve their design goals, and may be accomplished persons who are not necessarily specialists in information technology.

What do you monitor?

You can monitor service component events in WebSphere ESB by selecting certain points that a service component event reaches during processing. Each service component defines these event points, which generate (or "fire") an event when the application processes at that given point. You can also monitor performance statistics for service component events.

Regardless of the type of monitoring you intend to perform on your service components (problem determination, performance tuning, or process monitoring), you monitor a certain point that is reached during processing. This point is referred to as an *event point*, and it is these points that you select to be monitored. Each event point encapsulates the service component kind tag, an optional *element* kind (which are specific functions of a service component type), and the *nature* of the event. All these factors determine the type of event generated by monitoring.

Event natures describe the situations required to generate events during the processing of service components. These natures are key points in the logic structure of a service component that you select to be monitored. The most common natures for service component events are ENTRY, EXIT, and FAILURE, but there are many other natures depending on the particular component and element. Whenever an application containing the specified service component is later invoked, an event is fired every time the processing of a service component crosses the points corresponding to the event nature.

As an example of how events are defined for a service component kind, the MAP service component kind can directly fire events with natures of ENTRY, EXIT, and FAILURE. It also includes an element kind, called Transformation, which defines a specific type of functionality within the MAP component kind. This element also fires events with ENTRY, EXIT, and FAILURE natures. Consequently, the MAP service component kind can fire up to six different events depending on the combination of elements and natures that you specify. The list of all service components, their elements, and their event natures is contained in the event catalog.

Monitoring is a separate layer of functionality that lies atop the processing of your applications, and does not interfere with the processing of your service components. Monitoring is concerned with service component processing only insofar as it detects activity at a specified event point. When activity is detected, an

event is fired by monitoring, which determines where the event is sent, and what data is contained in that event, based on the type of monitoring you are performing:

Performance metrics

If you are monitoring a service component in order to gather performance metrics, light weight events are fired to the Performance Monitoring Infrastructure. You can select for monitoring one or more of the three performance statistics generated for server-specific server components:

- A counter for each EXIT event nature – counts successful computations.
- A counter for each FAILURE event nature – counts failed computations
- The processing duration calculated between corresponding ENTRY and EXIT events (synchronous computations only).

You can also monitor the performance of applications at the Service Component Architecture (SCA) level by using Application Response Measurement (ARM) statistics. These measures allow you to monitor an application at a much finer level of detail within the application than is otherwise available in other service component events. You can use these statistics to monitor many different points between initial application call invocations and service responses, when they use the SCA.

Service component events with business objects

If you want to capture the data from events fired by monitoring at specified event points in service component, then you would configure the server to generate the event and its data to be encoded in Common Base Event formats. You can specify the level of detail of business object data to capture in each service component event. You can publish these events to either a logger or to the Common Event Infrastructure (CEI) bus, which directs the output to a specially configured CEI server database.

How do you enable monitoring?

There are several methods that you can use to specify service component event points for monitoring, depending on the type of monitoring you are planning to do.

Performance statistics

For Performance Monitoring Infrastructure (PMI) statistics, use the administrative console to specify the particular event points and their associated performance measurements that you want to monitor. After you start monitoring service component performance, the generated statistics are published at certain intervals to the Tivoli Performance Viewer. You can use this viewer to watch the results as they occur on your system, and, optionally, log the results to a file that can be later viewed and analyzed within the same viewer.

For Application Response Measurement (ARM) statistics, use the administrative console Request Metrics section to specify and the statistics you want to monitor.

Common Base Events for problem determination and business process monitoring

You can specify, at the time you create an application, to monitor service component event points, along with a certain level of detail for those events, on a continual basis after the application is deployed on a running server. You can also select event points to monitor after the application has been deployed and the events invoked at least once. In both cases, the

events generated by monitoring are fired across the Common Event Infrastructure (CEI) bus. These events can be published to a log file, or to a configured CEI Server database. WebSphere ESB supports two types of Common Base Event enablement for problem determination and business process monitoring:

Static Certain events points within an application and their level of detail can be tagged for monitoring using Integration Designer tooling. The selections indicate what event points are to be continuously monitored, and are stored in a file with a .mon extension that is distributed and deployed along with the application. When WebSphere ESB has been configured to use a CEI server, the monitoring function begins firing service component events to a CEI server whenever the specified services are invoked. As long as the application is deployed on WebSphere ESB, the service component event points specified in the .mon file is constantly monitored until the application is stopped. You can specify additional events to be monitored in a running application, and increase the detail level for event points that are already monitored. But while that application remains active you cannot stop, or lower the detail level of, the monitored event points specified by the .mon of the deployed application.

Dynamic
If additional event points need to be monitored during the processing of an application without shutting down the server, then you can use dynamic monitoring. Use the administrative console to specify service component event points for monitoring, and set detail level for the payload that will be included in the Common Base Event. A list is compiled of the event points that have been reached by a processed service component after the server was started. Choose from this list individual event points or groups of event points for monitoring, with the service component events directed either to the logger or to the CEI server database.

The primary purpose of the Dynamic enablement is for creating correlated service component events that are published to logs, which allow you to perform problem determination on services. Service component events can be large, depending on how much data is being requested, and can tax database resources if you choose to send events to the CEI server. Consequently, you should publish dynamically monitored events to the CEI server only if you need to read the business data of the events, or if you otherwise need to keep a database record of the events. If, however, you are monitoring a particular session, then you need to use the CEI server database to access the service component events related to that session.

Related concepts:

"Monitoring performance" on page 965
Performance measurements are available for service component event points, and are processed through the Performance Monitoring Infrastructure. You configure the process server to gather performance metrics from service component event points. You can also collect Service Component Architecture-specific performance statistics directly from service invocations of applications.

"Session monitoring" on page 995
You can monitor multiple events that are part of the same session, by using the Common Base Event browser to find all events on the Common Event Infrastructure database that contain the identical session ID attribute.

Related tasks:

Chapter 10, "Enabling and configuring service component monitoring," on page 965
To be able to monitor service components, you must first enable the monitoring capabilities. Then you must specify the events you want to monitor, the information you want to capture from the event, and the method used to publish the results.

Administering Common Event Infrastructure

Getting performance data from request metrics

Chapter 10. Enabling and configuring service component monitoring

To be able to monitor service components, you must first enable the monitoring capabilities. Then you must specify the events you want to monitor, the information you want to capture from the event, and the method used to publish the results.

Related concepts:

"How do you enable monitoring?" on page 961
There are several methods that you can use to specify service component event points for monitoring, depending on the type of monitoring you are planning to do.

Monitoring performance

Performance measurements are available for service component event points, and are processed through the Performance Monitoring Infrastructure. You configure the process server to gather performance metrics from service component event points. You can also collect Service Component Architecture-specific performance statistics directly from service invocations of applications.

Whether you are tuning service components for optimal efficiency or diagnosing a poor performance, it is important to understand how the various run time and application resources are behaving from a performance perspective. The Performance Monitoring Infrastructure (PMI) provides a comprehensive set of data that explains the runtime and application resource behavior. Using PMI data, the performance bottlenecks in the application server can be identified and fixed. PMI data can also be used to monitor the health of the application server.

The PMI is included in the base WebSphere Application Server installation. This section provides only supplemental information about performance monitoring as it relates to the service components specific to WebSphere ESB; therefore, consult the information in the WebSphere Application Server documentation for using PMI with other parts of the entire product.

The service component event points specific to WebSphere ESB that can be monitored by the PMI are those events that include ENTRY, EXIT, and FAILURE event natures. Event sources which are not defined according to this pattern are not supported. Events that are supported have three types of performance statistics that can be measured:

- Successful invocations.
- Failed invocations.
- Elapsed time for event completion.

You can also monitor performance statistics derived from the service invocations of applications by using the Application Response Measurement (ARM) statistics. These statistics measure the actual runtime processes that underlie the process server service component events making up an enterprise application. You can derive various performance measurements for the processing of your applications using these statistics.

© Copyright IBM Corp. 2007, 2011

Related concepts:

"How do you enable monitoring?" on page 961
There are several methods that you can use to specify service component event points for monitoring, depending on the type of monitoring you are planning to do.

Performance Monitoring Infrastructure statistics

You can monitor three types of performance statistics using the Performance Monitoring Infrastructure: the number of successful invocations, the number of failures, and the elapsed time to completion of an event. These statistics are only available for events that have event natures of type ENTRY, EXIT, and FAILURE.

Enabling PMI using the administrative console

To monitor performance data you must first enable the Performance Monitoring Infrastructure on the server.

About this task

You can enable the Performance Monitoring Infrastructure (PMI) through the administrative console.

Procedure

1. Open the administrative console.
2. Click **Servers > Server Types > WebSphere application servers** in the console navigation tree.
3. Click *server_name*.

   Note: From the administrative console, you can click **Monitoring and Tuning > Performance Monitoring Infrastructure (PMI) >** *server_name* to open the same panel
4. Click the **Configuration** tab.
5. Select the **Enable Performance Monitoring Infrastructure (PMI)** check box.
6. Optional: Select the check box for **Use sequential counter updates** to enable precise statistic updates.
7. Go back to the server PMI configuration page by clicking the server name link.
8. Click **Apply** or **OK**.
9. Click **Save**.
10. Restart the server.

What to do next

The changes you make will not take effect until you restart the server.

Event performance statistics

Performance monitoring statistics are available for most server events. You can use performance monitoring statistics to monitor the counts of successful and unsuccessful invocation requests, and the time taken to complete events.

You can use the Performance Monitoring Infrastructure (PMI) to monitor three performance statistics generated by certain server events, as shown in the following table:

Table 84. PMI statistics for events

Statistic name	Type	Description
BadRequests	Counter	Number of failed invocations of the event.
GoodRequests	Counter	Number of successful invocations of the event.
ResponseTime	Timer	Elapsed time for event completion.

These statistics are limited to service component events with elements having ENTRY, EXIT, and FAILURE natures. Each statistic is created for a single event of a given server event type in an application. All performance measurements are either *counters* (a cumulative number of the firings of a given event point), or *timers* (the duration, measured in milliseconds, between the firings of two event points). Each event kind (and their relevant elements) that can be monitored are listed in Table 85:

Table 85. Event types and elements that can produce event performance statistics

Event type	Element(s)

Business process	`Process`
	`Invoke`
	`Staff`
	`Receive`
	`Wait`
	`Compensate`
	`Pick`
	`Scope`

Selector	`Operation`
Map	`Map` `Transformation`
Mediation	`OperationBinding` `ParameterMediation`

Resource adapter	`InboundEventRetrieval`
	`InboundEventDelivery`
	`Outbound`

Related reference:

"Application Response Measurement statistics for the Service Component Architecture" on page 971
There are 25 performance statistics that you can monitor at the Service Component Architecture (SCA) level. You can use these Application Response Measurement (ARM) statistics, which are either counters or timers, to measure invocations to and responses from services in various patterns.

Specifying performance statistics to monitor

You can specify single statistics, multiple statistics, or groups of related statistics for monitoring through the Performance Monitoring Infrastructure by using the administrative console.

Before you begin

Ensure that you have enabled performance monitoring, and that you have at least once invoked the event you want to monitor before performing this task.

Procedure

1. Open the administrative console.
2. Select **Monitoring and Tuning** > **Performance Monitoring Infrastructure**.
3. Select the server or node agent that contains the event points that you want to monitor.

   Note: You cannot choose to monitor statistics on a cluster; you can only do so on a specific server or node.
4. Expand some of the groups, such as WBIStats.RootGroup or Enterprise Beans. All the statistics that can be monitored are in the listed groups. Some statistics cannot be listed because they have not been invoked since the server was last started.
5. Select a statistic you want to monitor from the tree, and then select the statistics that you want to collect. Then click **Enable**. Repeat for all statistics that you want to monitor.
6. Go back to the server PMI configuration page by clicking the server name link.
7. Click **Apply** or **OK**.
8. Click **Save**.

Results

You can now start monitoring the performance of your chosen statistics in the Tivoli Performance Viewer.

Note: When viewing these statistics, Do not mix counter-type statistics with duration-type statistics. Counters are cumulative, and the scales against which they are graphed them can quickly grow depending on your application. Duration statistics, in contrast, tend to remain within a certain range because they represent the average amount of time that it takes your system to process each event. Consequently, the disparity between the statistics and their relative scales can cause one or the other type of statistic to appear skewed in the viewer graph.

Tutorial: Service component performance monitoring

This tutorial guides you through an example of setting up performance monitoring, and how to view the resulting statistics.

For service component event points that you monitor, you can publish to the Performance Monitoring Infrastructure (PMI) and view the resulting performance statistics on the Tivoli Performance Viewer (TPV). This exercise demonstrates how performance monitoring of service component event points differs from monitoring using the Common Event Infrastructure (CEI) server and loggers. The major difference that you notice is that you select an entire service component element for performance monitoring, instead of individual events with specific natures. Because WebSphere ESB can monitor performance only on service component elements having events with ENTRY, EXIT, and FAILURE natures, you have only those kinds of service component elements available to you to select for monitoring.

While the service component event points ENTRY, EXIT, and FAILURE are identical for all monitoring types, the performance monitoring function in the server fires "minimized" events that do not contain all the information encompassed in CEI events. These events are sent to the PMI, which calculates these performance statistics from corresponding sets of events:

- Successful invocation — the firing of an event of nature type EXIT that follows a corresponding ENTRY event.
- Failed invocation — the firing of an event with a FAILURE nature following a corresponding ENTRY event.
- Time for successful completion — the elapsed time between the firing an ENTRY event and the firing of the corresponding EXIT event point.

The PMI publishes the statistics to the TPV, which presents cumulative counters for the number of successful and failed invocations and a running average of the completion response times.

Objectives of this tutorial

After completing this tutorial, you will be able to:

- Select the performance statistics of service component elements that you want to monitor.
- View and interpret the resulting performance statistics.

Time required to complete this tutorial

This tutorial requires approximately 15-20 minutes to complete.

Prerequisites

In order to perform this tutorial, you must have:

- Configured and started a server.
- Enabled the PMI on the server.
- Installed and started the Samples Gallery application on the server.
- Installed and started the business rules sample application on the server. Follow the instructions on the Samples Gallery page to set up and run the business rules sample application.

After all these prerequisites have been completed, run the business rules sample application from the Samples Gallery at least once before proceeding with the tutorial.

Example: Monitoring service component performance:

For monitoring performance, you can use the administrative console to select service components for monitoring and view performance measurements. This example shows the use of the console to monitor performance statistics.

About this task

You will use the business rules sample application for this scenario, where you will monitor all three of the performance statistics: successes, failures, and response times. You should have the web page containing this application already open; keep it open, because you will be running the sample several times after you begin monitoring. Ensure that you have already run the sample at least once, which causes it to appear in the list of functions that you can select to monitor.

Procedure

1. Open the administrative console.
2. Select the cluster or server to monitor.

- To monitor a cluster, click **Servers** > **Clusters** > **WebSphere application server clusters** > *cluster_name*.
- To monitor a single server, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name*.

3. Click the Runtime tab.
4. Under Performance, click **Performance Monitoring Infrastructure**.
5. Select **Custom**.
6. Expand **WBIStats.RootGroup** > **BR** > **brsample_module.DiscountRuleGroup** > **Operation**.
7. Select **_calculateDiscount**
8. Select the check boxes next to BadRequests, GoodRequests, and ResponseTime.
9. Click **Enable**
10. In the navigation pane, click **Monitoring and Tuning** > **Performance Viewer** > **Current Activity**.
11. Select the check box next to *server_name*, then click **Start Monitoring**.
12. Click *server_name*.
13. Expand **WBIStats.RootGroup** > **BR** > **brsample_module.DiscountRuleGroup** > **Operation**.
14. Select the check box next to _calculateDiscount

Results

You should now see a blank graph, and underneath that the names and values for the three statistics. Select the check boxes next to the statistic names, if they are not already checked. The PMI is now ready to publish performance data for the selected event, and the Tivoli Performance Viewer is ready to present the results.

Run the business rules sample application several times, and then watch the performance viewer as it periodically refreshes. Notice that there are now lines on the graph, representing the cumulative number of successful requests and the average response time for each successful request. You can also see the values next to the name for each statistic below the graph. The line for the number of successes should continue to rise as you perform additional invocations of the sample, while the response time line should level off after a few refreshes.

After you have completed this example, you should understand how WebSphere ESB implements performance monitoring of service components. You should know how to select service components for monitoring, and how the performance statistics are calculated. You will also be able to start the performance monitors, and view the performance measurements for your applications as they are being used.

What to do next

Performance monitoring can tax system resources; therefore, after you have completed this task you should stop the monitors. To do this, click the Tivoli Performance Viewer link, select both the node and the server, and press **Stop Monitoring**.

Application Response Measurement statistics for the Service Component Architecture

There are 25 performance statistics that you can monitor at the Service Component Architecture (SCA) level. You can use these Application Response Measurement (ARM) statistics, which are either counters or timers, to measure invocations to and responses from services in various patterns.

The Application Response Measurement (ARM) statistics shown in the following tables are — in a simplified manner — time and count measurements of caller invocations to the Service Component Architecture (SCA) layer, and the results returned from a service. There are, in fact, a number of service invocation patterns that vary between synchronous and asynchronous implementations of deferred responses, results retrievals, callbacks, and one-way invocations. All patterns, however, are between the caller invocation and a service, the response from the service, or, in some cases, a data source, with the SCA layer interposed in between.

You can specify the ARM statistics that you want to monitor by opening the **Monitoring and Tuning > Request Metrics** panel on the administrative console. Request metrics information might be either saved to the log file for later retrieval and analysis, be sent to ARM agents, or both. WebSphere ESB does not ship an ARM agent; however, it supports the use of agents adhering to ARM 4.0. You can choose your own ARM implementation provider to obtain the ARM implementation libraries. Follow the instructions from the ARM provider, and ensure that the ARM API Java archive (JAR) files found in the ARM provider are on the class path so that WebSphere ESB can load the needed classes. Then you need to add the following entries into the system properties for each server by selecting from the administrative console **Application servers >** ***server_name*** **> Process Definition > Java Virtual Machine > Custom Properties** before restarting the server:

- `Arm40.ArmMetricFactory` — the full Java class name of your ARM implementation providers metrics factory.
- `Arm40.ArmTranReportFactory` — the full Java class name of your ARM implementation providers transaction report factory.
- `Arm40.ArmTransactionFactory` — the full Java class name of your ARM implementation providers transaction factory.

See the WebSphere Application Server documentation for further details on how to configure the server to collect ARM statistics.

Table 86. Event types and elements that can produce ARM statistics

Event type	Element
Selector	`Operation`
Map	`Map` `Transformation`
Mediation	`OperationBinding` `ParameterMediation`

Resource adapter	`InboundEventRetrieval`
	`InboundEventDelivery`
	`Outbound`

Table 87. **Common**. These statistics are common to all service invocation patterns.

Statistic name	Type	Description
GoodRequests	Counter	Number of server invocations not raising exceptions.
BadRequests	Counter	Number of server invocations raising exceptions.
ResponseTime	Timer	Duration measured on the server side between the reception of a request and computing the result.
TotalResponseTime	Timer	Duration measured on the caller side, from the time a caller requests a service to the time when the result is available for the caller. Does not include the processing of the result by the caller.
RequestDeliveryTime	Timer	Duration measured on the caller side, from the time a caller requests a service to the time when the request is handed over to the implementation on the server side. In a distributed environment, the quality of this measurement depends on the quality of synchronization of system clocks.
ResponseDeliveryTime	Timer	The time required to make the result available to the client. For a deferred response, this time does not include the result retrieve time. In a distributed environment, the quality of this measurement depends on the quality of synchronization of system clocks.

Table 88. **Reference**. These statistics occur when a caller makes an invocation to the SCA layer or a data source, without a response from the service.

Statistic name	Type	Description
GoodRefRequests	Counter	Number of caller invocations to the SCA layer that do not raise exceptions.
BadRefRequests	Counter	Number of caller invocations to the SCA layer that do raise exceptions.
RefResponseTime	Timer	Duration measured on the caller side, from the time the caller makes a request to the SCA layer and the time when the results of that call are returned to the caller.
BadRetrieveResult	Counter	Number of caller invocations to a data source that do raise exceptions.
GoodRetrieveResult	Counter	Number of caller invocations to a data source that do not raise exceptions.
RetrieveResultResponseTime	Timer	Duration measured on the caller side, from the time the caller makes a request to the data source and the time when the data source response is returned to the caller.
RetrieveResultWaitTime	Timer	Duration measured on the caller side if a timeout occurs.

Table 89. **Target**. These statistics occur when there are requests that originate between the service and the SCA or a data source.

Statistic name	Type	Description
GoodTargetSubmit	Counter	Number of SCA invocations to the service that do not raise exceptions.
BadTargetSubmit	Counter	Number of SCA invocations to the service that do raise exceptions.

Table 89. **Target** *(continued).* These statistics occur when there are requests that originate between the service and the SCA or a data source.

Statistic name	Type	Description
TargetSubmitTime	Timer	Duration measured on the server side, from the time the SCA makes a request to the service and the time when the results of that call are returned to the SCA.
GoodResultSubmit	Counter	Number of service invocations to the data source that do not raise exceptions.
BadResultSubmit	Counter	Number of service invocations to the data source that do raise exceptions.
ResultSubmitTime	Timer	Duration measured on the server side, from the time the service makes a request to the data source and the time when the results of are returned to the service.

Table 90. **Callback**. These statistics occur when a callback (a "sibling" of the original call) is present on the caller.

Statistic name	Type	Description
GoodCB	Counter	Number of SCA invocations to the callback that do not raise exceptions.
BadCB	Counter	Number of SCA invocations to the callback that do raise exceptions.
CBTime	Timer	Duration from the time the SCA makes a request to the callback, and the time when the results from the callback are returned to the SCA.
GoodCBSubmit	Counter	Number of invocations from the service to the SCA handling the callback that do not raise exceptions.
BadCBSubmit	Counter	Number of invocations from the service to the SCA handling the callback that do raise exceptions.
CBSubmitTime	Timer	Duration from the time the service makes a request to the SCA handling the callback, and the time when the results from the SCA to the service.

Related reference:

Performance Monitoring Infrastructure statistics
This section tells you about different performance statistics in WebSphere ESB.

Related information:

WebSphere Application Server documentation Network Deployment

Synchronous invocations

You can obtain Application Response Measurement (ARM) performance statistics from a simple Service Component Architecture (SCA) call to a service and the response from the service.

Parameters

Event monitoring for SCA components includes the event points that are shown in black , while the event points shown in blue are used only to calculate and fire PMI/ARM statistics.

In Table 91 on page 974 and Figure 309 on page 974, the “current” ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction is used, or a new one is created. If it is not the starting transaction then it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$.

The notation is used to document the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but they do not modify the SCA transaction lineage.

Table 91. ARM statistics for synchronous invocations of SCA

Statistics	Formula	ARM Transaction
TotalResponseTime	$t_3 - t_0$	$X_0 . X_1$
RequestDeliveryTime	$t_1 - t_0$	$X_1 . X_2$
ResponseDeliveryTime	$t_3 - t_2$	
GoodRequests	$Count_{EXIT}$	
BadRequests	$Count_{FAILURE}$	
ProcessTime	$t_2 - t_1$	

Figure 309. ARM statistics obtained from an SCA call with a synchronous implementation

Deferred response with synchronous implementation

You can obtain Application Response Measurement (ARM) statistics with a synchronous invocation of the request. The returned result is sent as output to a data store for a synchronous implementation.

Parameters

Event monitoring for Service Component Architecture (SCA) components includes the event points that are shown in black , while the event points shown in blue are used only to calculate and fire PMI/ARM statistics.

In Table 92 on page 975 and Figure 310 on page 975, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction is used, or a new one is created. If it is not the starting transaction, it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$. The notation is used to show the transaction lineage. Every SCA invocation starts a

new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but you cannot modify the SCA transaction lineage.

Table 92. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_3 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	$Count_{EXIT}$	$X_1.X_2$
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	$t'_1 - t'_0$	
Reference A	GoodRefRequest	$Count_{EXIT}$	$X_1.X_2$
	BadRefRequests	$Count_{FAILURE}$	
	RefResponseTime	$t_1 - t_0$	

Figure 310. Graphic representation of a deferred response with synchronous implementation

Table 93. Invocation of output to data source

Type	Statistics	Formula	ARM Transaction
Reference B	GoodRetrieveResult	$Count_{EXIT}$	$X_1.X_2$
	BadRetrieveResult	$Count_{FAILURE}$	
	ResultRetrieveResponseTime	$\Sigma\ t_3 - t_2$	
	ResultRetrieveWaitTime	Σ timeout	

Deferred response with asynchronous implementation

You can obtain Application Response Measurement (ARM) statistics from an asynchronous implementation. The call to the service and the return result are invoked but the resulting output is sent to a data store from the service target.

Parameters

Event monitoring for Service Component Architecture (SCA) components includes the event points that are shown in black , while the event points shown in blue are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction is used, or a new one is created. If it is not the starting transaction, it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$. The notation is used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but you cannot modify the SCA transaction lineage.

Table 94. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_3 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	$t'_{03} - t'_2$	
	GoodRequests	$Count_{EXIT}$	
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	$t'_3 - t'_0$	
Reference A	GoodRefRequest	$Count_{EXIT}$	$X_0.X_1$
	BadRefRequests	$Count_{FAILURE}$	
	RefResponseTime	$t_1 - t_0$	
Target A	GoodTargetSubmit	$Count_{EXIT}$	$X_1.X_2$
	BadTargetSubmit	$Count_{FAILURE}$	
	TargetSubmitTime	$t'_1 - t'_0$	

Table 95. Invocation of return result to a data store

Type	Statistics	Formula	ARM Transaction
Reference B	GoodResultSubmit	$Count_{EXIT}$	$X_0.X_1$
	BadResultSubmit	$Count_{FAILURE}$	
	ResultResponseTime	$t'_3 - t'_2$	
Target B	GoodResultRetrieve	$Count_{EXIT}$	$X_1.X_2$
	BadResultRetrieve	$Count_{FAILURE}$	
	ResultRetrieveResponseTime	$\Sigma\ t_3 - t_2$	
	ResultRetrieveWaitTime	Σ timeout	

Deferred response with asynchronous result retrieve

The ResultRetrieve Application Response Measurement (ARM) statistic can be correlated to some original request using the ARM transactions only if $X_{PARENT-1}$ and $X_{PARENT-2}$ have a common ancestor transaction. The invocation of request, and result retrieve occur on different threads

Parameters

Event monitoring for Service Component Architecture (SCA) components includes the event points that are shown in black , while the event points shown in blue are used only to calculate and fire PMI/ARM statistics.

In Table 96 and Figure 311 on page 979, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction will be used, or a new one will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n.X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 96. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_3 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	$Count_{EXIT}$	$X_1.X_2$
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	See specific diagrams	
Reference A	GoodReferenceRequest	$Count_{EXIT}$	$X_1.X_2$
	BadReferenceRequests	$Count_{FAILURE}$	
	ReferenceResponseTime	$t_1 - t_0$	

Figure 311. A deferred response with an asynchronous result retrieve

Table 97. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Reference B	GoodRetrieveResult	$Count_{EXIT}$	$X'_0.X'_1$
	BadRetrieveResult	$Count_{FAILURE}$	
	RetrieveResultResponseTime	$\Sigma\ t_3 - t_2$	
	RetrieveResultWaitTime	Σ timeout	

Asynchronous callback with synchronous implementation

You can obtain Application Response Measurement (ARM) statistics when callback requests and callback executions use different threads on a synchronous implementation.

Parameters

Event monitoring for Service Component Architecture (SCA) components includes the event points that are shown in black , while the event points shown in blue are used only to calculate and fire PMI/ARM statistics.

In Table 98 on page 980 and Figure 312 on page 980, the “current” ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction is used, or a new one is created. If it is not the starting transaction, it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$. The notation is used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but you cannot modify the SCA transaction lineage.

Table 98. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_2 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	$t_2 - t'_1$	
	GoodRequests	$Count_{EXIT}$	
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	$t_3 - t_2$	
Reference	GoodRefRequest	$Count_{EXIT}$	$X_1.X_2$
	BadRefRequests	$Count_{FAILURE}$	
	RefResponseTime	$t'_1 - t'_0$	

Figure 312. Diagram of an asynchronous callback with a synchronous implementation

Table 99. Invocation of callback

Type	Statistics	Formula	ARM Transaction
Callback	GoodCB	$Count_{EXIT}$	$X_1.X_3$
	BadCB	$Count_{FAILURE}$	
	CBTime	$t_3 - t_2$	

Asynchronous callback with asynchronous implementation

Application Response Measurement (ARM) statistics are available for callback requests and callback executions using different threads with an asynchronous implementation

Parameters

Event monitoring for Service Component Architecture (SCA) components includes the event points that are shown in black , while the event points shown in blue are used only to calculate and fire PMI/ARM statistics.

In Table 100 and Figure 313 on page 982, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction is used, or a new one is created. If it is not the starting transaction it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$. The notation is used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but you cannot modify the SCA transaction lineage.

Table 100. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_2 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	$t_2 - t'_2$	
	GoodRequests	$Count_{EXIT}$	
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	$t'_3 - t'_0$	
Reference A	GoodRefRequest	$Count_{EXIT}$	$X_0.X_1$
	BadRefRequests	$Count_{FAILURE}$	
	RefResponseTime	$t_1 - t_0$	
Target A	GoodTargetSubmit	$Count_{EXIT}$	$X_1.X_2$
	BadTargetSubmit	$Count_{FAILURE}$	
	TargetSubmitTime	$t'_1 - t'_0$	

Figure 313. An asynchronous callback with an asynchronous implementation

Table 101. Invocation of callback

Type	Statistics	Formula	ARM Transaction
Reference B	GoodCBSubmit	$Count_{EXIT}$	$X_1.X_2$
	BadCBSubmit	$Count_{FAILURE}$	
	CBSubmitTime	$t'_3 - t'_2$	
Target B	GoodCB	$Count_{EXIT}$	$X_0.X_1$
	BadCB	$Count_{FAILURE}$	
	CBTime	$t_3 - t_2$	

Asynchronous one way with synchronous implementation

These Application Response Measurement (ARM) statistics can be obtained when a call is submitted (fire and forget) with a synchronous implementation.

Parameters

Event monitoring for Service Component Architecture (SCA) components includes the event points that are shown in black , while the event points shown in blue are used only to calculate and fire PMI/ARM statistics.

In Table 102 and Figure 314, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction is used, or a new one is created. If it is not the starting transaction, it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$. The notation is used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but you cannot modify the SCA transaction lineage.

Table 102. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_1 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	$Count_{EXIT}$	$X_1.X_2$
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	$t'_1 - t'_0$	

Figure 314. Diagram of asynchronous one-way call with a synchronous implementation

Asynchronous one way with asynchronous implementation

Application Response Measurement (ARM) statistics when a call is submitted (fire and forget) with an asynchronous implementation.

Parameters

Event monitoring for Service Component Architecture (SCA) components includes the event points that are shown in black , while the event points shown in blue are used only to calculate and fire PMI/ARM statistics.

In Table 103 on page 984 and in Figure 315 on page 984, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction is used, or a new one is created. If it is not the starting transaction, it has a parent. This relationship is represented in the following table and diagram

with the notation $X_n.X_{n+1}$. The notation is used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but you cannot modify the SCA transaction lineage.

Table 103. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_1 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	$Count_{EXIT}$	$X_1.X_2$
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	$t_2 - t_0$	
Reference	GoodRefRequest	$Count_{EXIT}$	$X_0.X_1$
	BadRefRequest	$Count_{FAILURE}$	
	RefResponseDuration	$t_1 - t_0$	

Figure 315. An asynchronous one-way call with an asynchronous implementation

Monitoring service component events

WebSphere ESB monitoring can capture the data in a service component at a certain event point. You can view each event in a log file, or you can use the more versatile monitoring capabilities of a Common Event Infrastructure server.

Applications that are deployed on the process server may contain a specification of service component events that will be monitored for as long as the application runs. If you developed the application using the WebSphere Integration Developer, then you can specify service component events to monitor continuously. This specification is included as part of the application, and comes in the form of file with a .mon extension that is read by the process server when the application is deployed. After the application is started, you will not be able to turn off monitoring of the service components specified in the .mon file. The

documentation for the WebSphere ESB does not address this type of continuous monitoring. For more information about this subject, refer to the Integration Designer documentation.

You can use WebSphere ESB to monitor service component events that are not already specified in the .mon file of the application. You can configure the process server to direct the output of the event monitors to a log file, or to a Common Event Infrastructure server database. The monitored events will be formatted using the Common Base Event standard, but you can regulate the amount of information contained in each event. Use the monitoring facilities in WebSphere ESB to diagnose problems, analyze the process flow of your applications, or audit how your applications are used.

Enabling monitoring of business process and human task events

You must configure WebSphere ESB to support monitoring of business process and human task service components before you do any actual monitoring of those service component kinds.

Before you begin

You must have previously created the business process container and the human task container on the process server.

About this task

Perform this task to enable Common Event Infrastructure monitoring support on WebSphere Process Server.

Procedure

1. Open the administrative console.
2. If Business Process Choreographer is configured on a single server, complete the following steps to enable the server to generate business process events:
   a. To enable business process events for the Human Task Manager, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name*, then on the **Configuration** tab under **Business Integration**, expand **Business Process Choreographer**, click **Human Task Manager**. In the section **State Observers**, ensure that the boxes for **Enable Common Event Infrastructure Logging**, **Enable audit logging**, and **Enable task history** are selected. If the check boxes are not selected, then you must select them and restart the server.
   b. To enable business process events for the Business Flow Manager, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name*, then on the **Configuration** tab under **Business Integration**, expand **Business Process Choreographer**, click **Business Flow Manager**. In the section **State Observers**, ensure that the boxes for **Enable Common Event Infrastructure Logging** and **Enable audit logging** are selected. If the check boxes are not selected, then you must select them and restart the server.
3. If Business Process Choreographer is configured on a cluster, complete the following steps to enable the cluster to generate business process events:
   a. To enable business process events for the Human Task Manager, click **Servers** > **Clusters** > **WebSphere application server clusters** > *cluster_name*, then on the **Configuration** tab under **Business Integration**, expand **Business Process Choreographer**, click ensure that the boxes for

Enable Common Event Infrastructure Logging, **Enable audit logging**, and **Enable task history** are selected. If the check boxes are not selected, then you must select them and restart the server.

b. To enable business process events for the Business Flow Manager, click **Servers** > **Clusters** > **WebSphere application server clusters** > ***cluster_name***, then on the **Configuration** tab under **Business Integration**, expand **Business Process Choreographer**, click **Business Flow Manager**. In the section **State Observers**, ensure that the boxes for **Enable Common Event Infrastructure Logging** and **Enable audit logging** are selected. If the check boxes are not selected, then you must select them and restart the server.

What to do next

If you had to select any of the boxes, then you must restart the server or cluster for the changes to take effect.

Configuring logging for service component events

You can choose to use the logging facilities of WebSphere Application Server to capture the service component events fired by process server monitoring. Use the loggers to view the data in events when you diagnose problems with the processing of your applications.

WebSphere ESB uses the extensive logging facilities of the underlying WebSphere Application Server to allow you to capture the events fired by server monitoring at service component event points. You can use the administrative console to specify the particular service component event points that you want to monitor, the amount of payload detail contained in the resulting service component events, and the method used to publish the results, such as to a file of a certain format, or directly to a console. Monitor logs contain events encoded in Common Base Event format, and you can use the information contained in the event elements to trace problems with the processing of your service components.

The functionality of WebSphere Application Server logging and tracing capabilities is documented in considerable detail in the WebSphere Application Server documentation, with complete details of how logging and tracing is used within the entire product. This section provides only supplemental information about logging as it relates to the service components that are specific to WebSphere ESB. Consult the information in the WebSphere Application Server documentation for using logging and trace with other components of the entire product.

Enabling the diagnostic trace service

Use this task to enable the diagnostic trace service, which is the logging service that can manage the amount of detail contained in the service component event.

About this task

The diagnostic trace service is the only logger type that can provide the level of detail required to capture the detail contained in the elements of service component events. You must enable the diagnostic trace service before you start the process server in order to log events. The service must also be enabled if you use the administrative console to select service component event points for monitoring using the CEI server.

Procedure

1. In the navigation pane, click **Servers** > **Server Types** > **WebSphere application servers**.
2. Click the name of the server that you want to work with.
3. Under Troubleshooting, click **Diagnostic Trace service**.
4. Select **Enable log** on the Configuration tab.
5. Click **Apply**, and then **Save**.
6. Click **OK**.

What to do next

If the server was already started, then you must restart it for the changes to take effect.

Configuring logging properties using the administrative console

Use this task to specify that the monitoring function publish service component events to a logger file.

About this task

Before applications can log monitored events, you must specify the service component event points that you want to monitor, what level of detail you require for each event, and format of the output used to publish the events to the logs. Using the administrative console, you can:

- Enable or disable a particular event log.
- Specify the level of detail in a log.
- Specify where log files are stored, how many log files are kept, and a format for log output.

You can change the log configuration statically or dynamically. Static configuration changes affect applications when you start or restart the application server. Dynamic or run time configuration changes apply immediately.

When a log is created, the level value for that log is set from the configuration data. If no configuration data is available for a particular log name, the level for that log is obtained from the parent of the log. If no configuration data exists for the parent log, the parent of that log is checked, and so on, up the tree until a log with a non-null level value is found. When you change the level of a log, the change is propagated to the children of the log, which recursively propagates the change to their children, as necessary.

Procedure

1. Enable logging and set the output properties for a log:
2. In the navigation pane, click **Servers > Server Types > WebSphere application servers**.
3. Click the name of the server that you want to work with.
4. Under Troubleshooting, click **Logging and tracing**.
5. Click **Change Log Detail levels**.
6. The list of components, packages, and groups displays all the components that are currently registered on the running server; only server events that have been invoked at least once appear on this list. All server components with

event points that can be logged are listed under one of the components that start with the name **WBILocationMonitor.LOG.**

- To select events for a static change to the configuration, click the Configuration tab.
- To select events for a dynamic change to the configuration, click the Runtime tab.

7. Select the event or group of events that you want to log.
8. Set the logging level for each event or group of events.

   Note: Only the levels FINE, FINER, and FINEST are valid for CEI event logging.
9. Click **Apply**.
10. Click **OK**.
11. To have static configuration changes take effect, stop then restart the server.

Results

By default, the loggers publish their output to a file called trace.log, located in the `install_root/profiles/profile_name/logs/server_name` folder.

Tutorial: Logging service component events

For service component event points that you monitor, events can be published to the logging facilities of the underlying WebSphere Application Server. This tutorial guides you through an example of setting up monitoring with logging, and how to view events stored in a log file.

The scenario you will follow for this example will show you how to select service component event points for monitoring in applications already deployed and running on a server. You will see how the monitoring function fires an event whenever the processing of an application reaches one of those event points. Each of those fired events takes the form of a standardized Common Base Event, which is published as an XML string directly to a log file.

Objectives of this tutorial

After completing this tutorial you will be able to:

- Select service component event points to monitor, with the output published to the server loggers.
- View the stored events in the log files.

Time required to complete this tutorial

This tutorial requires approximately 15-20 minutes to complete.

Prerequisites

In order to perform this tutorial, you must have:

- Configured and started a server.
- Configured Common Event Infrastructure.
- Enabled the diagnostic trace service on the server.
- Installed and started the Samples Gallery application on the server.

- Installed and started the business rules sample application on the server. Follow the instructions on the Samples Gallery page to set up and run the business rules sample application.

After all of these prerequisites have been completed, run the business rules sample application from the Samples Gallery at least once before proceeding with the tutorial.

Example: Monitoring events in the logger:

For monitoring with logging, you can use the administrative console to manage the details for event types. This example shows the use of the console to change the level of detail recorded for some event types and to use a text editor to open the trace.log file to view the information for individual events.

About this task

You will use the business rules sample application for this scenario, so you should already have the web page containing this application already open. Keep it open, since you will be running the sample after you specify monitoring parameters. Ensure that you have already run the sample at least once, so that it will appear in the list of functions that you can select to monitor.

Procedure

1. Open the administrative console.
2. In the navigation pane, click **Servers** > **Application Servers**.
3. Click *server_name*.
4. Under Troubleshooting, click **Logging and tracing**
5. Click **Change Log Detail levels**
6. Select the **Runtime** tab.
7. Expand the tree for **WBILocationMonitor.LOG.BR** and you will see seven event types under the **WBILocationMonitor.LOG.BR.brsample.*** element. The first event is called **WBILocationMonitor.LOG.BR.brsample_module.DiscountRuleGroup**, which includes a single function, named **Operation._calculateDiscount**, with the following natures:
   - ENTRY
   - EXIT
   - FAILURE
   - SelectionKeyExtracted
   - TargetFound
8. Click on each of the events and select **finest**.
9. Click **OK**.
10. Switch the business rules sample application page, and run the application once.
11. Use a text editor to open the trace.log file located in the `profile_root/logs/server_name` folder on your system.

Results

You should see lines in the log containing the business rule events fired by the monitor when you ran the sample application. The main thing you will probably notice is that the output consists of lengthy, unparsed XML strings conforming to

the Common Base Event standard. Examine the ENTRY and EXIT events, and you will see that business object — which was included because you selected the **finest** level of detail — is encoded in hexadecimal format. Compare this output with events published to the Common Event Infrastructure server, which parses the XML into a readable table and decodes any business object data into a readable format. You may want to go back through this exercise and change the level of detail from **finest** to **fine** or **finer**, and compare the differences between the events.

After completing this exercise, you should understand how to select service component event points for monitoring to the logger. You have seen that the events fired in this type monitoring have a standard format, and that the results are published as a string in raw XML format directly to a log file. To view the published events, open the log file in a text editor, and decipher the contents of individual events.

What to do next

If you no longer want to monitor the business rules sample application, you can go back to through the steps outlined here and reset the level of detail for the sample events to **info**.

Audit logging for business rules and selectors

You can set up WebSphere ESB to automatically log any changes made to business rules and selectors.

You can configure your server to automatically detect when changes are made to business rules and selectors, and create an entry in a log file detailing the changes.

You can choose to have the log entries written to either the standard JVM SystemOut.log file, or to a custom audit log file of your choice. Depending on how the changes are made, the process server where each business rule or selector change is made logs the:

- name of the person making the change
- location from where the change request originated
- old business rule or selector object
- new business rule or selector replacing the old object

The business rule and selector objects are the complete business rule set, decision table, business rule group, or selector for both the business rule or selector that is replaced and the new version which replaced it. You can examine the logs (the audit output cannot be directed to the Common Event Infrastructure database) to determine the changes that were made, by comparing the old and new business rules or selectors. The following scenarios describe the circumstance when logging occurs, if it has been configured, and the contents of the log entry:

Scenario	Result	Log entry contents
Publish business rules using the Business Rule Manager	Request	User ID, Server name (including Cell and Node, if applicable), old business rule ruleset, new ruleset.
	Failure	User ID, Server name (including Cell and Node, if applicable), old business rule ruleset, new ruleset.

Scenario	Result	Log entry contents
Repository database update and commit (from attempt to publish using the Business Rule Manager)	Success	User ID, old ruleset, new ruleset.
	Failure	User ID, new ruleset.
Exporting a selector or business rule group	Request	User ID, selector, or business rule group name.
	Success	User ID, Server name (including Cell and Node, if applicable), copy of exported selector or business rule group
	Failure	User ID, Server name (including Cell and Node, if applicable), selector or business rule group name.
Importing a selector or business rule group	Request	User ID, copy of new selector or business rule group.
	Success	User ID, Server name (including Cell and Node, if applicable), copy of imported selector or business rule group, copy of selector or business rule group that was replaced by the imported version.
	Failure	User ID, Server name (including Cell and Node, if applicable), copy of selector or business rule group that was to be imported.
Application installation	Success	User ID, Server name (including Cell and Node, if applicable), selector or business rule group name.
	Failure	User ID, Server name (including Cell and Node, if applicable), selector or business rule group name.
Application update (through the administrative console or **wsadmin** command)	Success	User ID, Server name (including Cell and Node, if applicable), copy of new selector or business rule group, copy of old selector or business rule group.
	Failure	User ID, Server name (including Cell and Node, if applicable), copy of new selector or business rule group
Previously deployed application with existing business rules, selectors or both is started	Success	Server name (including Cell and Node, if applicable), copy of selector or business rule group.
	Failure	Server name (including Cell and Node, if applicable), copy of selector or business rule group.

Monitoring service components with the Common Event Infrastructure server

You can choose to have service component monitoring results published to a Common Event Infrastructure (CEI) server. Service component event points can be specified for monitoring with the CEI server on a permanent basis for viewing and managing application flow, or on an temporary basis for troubleshooting problems.

You can use monitoring to publish the data in service component event points within service component events that are fired across the CEI bus. This approach to monitoring allows you much more flexibility in analyzing your service component activities on your system. You can also use browsers optimized for CEI events, such as the Common Base Event browser.

The events are structured identically to the events sent to loggers, but are stored on a database which can be accessed by viewers designed specifically for analyzing service component events. Service component event points can be specified within an application when it is created, for continual monitoring at all times after the application is deployed and running on a server (a method known as "static" monitoring). You perform static monitoring on service component event points that are of particular importance in the proper flow of component processing on your system. With this information, you can easily oversee the actions of, and interactions between, the service component processes running on your system. You can also quickly detect deviations from the normal flow of these processes, which can indicate that your service components are not working properly.

To configure static monitoring of service components, you use Integration Designer to select the service component event points in your applications. The selections are specified in the form of an XML file with a .mon extension that is deployed along with the application. After you have deployed the application on a running server, you cannot turn off or lower the detail level of the monitoring for events specified in the .mon file of the application. To stop this monitoring, you must stop the server and undeploy the application.

You can also select service component event points for "dynamic" monitoring, which can be enabled and disabled on an application already deployed to a running server. The rationale for performing dynamic monitoring using the CEI server is essentially the same as that for logging: to diagnose and troubleshoot problems on your system. The output is essentially the same as the output that is published to loggers, with Common Base Event elements that make up the structure for each event fired across the CEI bus. Also, like logging data, the differences in detail levels affect only how much of the payload is encoded within the event.

Configuring service component event monitoring using the administrative console

Use the administrative console to dynamically specify the monitoring function to publish service component events to the Common Event Infrastructure server.

Before you begin

You must enable the diagnostic trace service, just as you would with the logger. After you restart your server you would invoke the events you want to monitor once, because that will cause them to appear on the list of events available for monitoring.

About this task

This method of selecting events for monitoring is used for applications that have already been deployed on a process server. Events that are specified in a .mon file that is deployed with the application on the process server are monitored by the Common Event Infrastructure (CEI) database regardless of any changes you make here. For those events, you can only specify a greater level of detail to be captured and published to the CEI database. The output that is published to the CEI

database is very similar to that published by loggers.

Procedure

1. From the administrative console, click **Troubleshooting > Logging and tracing**.
2. Click **Change Log Detail levels**
3. The list of components, packages, and groups displays all the components that are currently registered on the running server; only process server events that have been invoked at least once appear on this list. All process server events that can be logged are listed under one of the components that start with the name **WBILocationMonitor.CEI**.
   - To make a static change to the configuration, click the **Configuration** tab.
   - To change the configuration dynamically, click the **Runtime** tab.
4. Select an event or group of events to monitor.
5. Click the level of detail that you want to capture for each event.

   Note: Only the levels FINE, FINER, and FINEST are valid for CEI events.
6. Click **Apply**, and then **Save**.
7. Click **OK**.
8. If you made a static change to the configuration, then you will have to restart the process server for the changes to take effect.

Results

You can view the monitored event results in the Common Base Event browser.

Tutorial: Using the Common Event Infrastructure server for event monitoring

This tutorial guides you through an example of setting up monitoring with the CEI server, and how to view events stored in the database.

For service component event points that you monitor, events can be published to the Common Event Infrastructure (CEI) server and stored in the CEI server database. Once events have been captured, use the Common Base Event browser to view those stored events. The example you use in this scenario does not involve static monitoring, whereby an application deployed with a .mon file continually monitors specific service components event points. For information about how to perform static monitoring, consult the IBM Integration Designer Information Center.

The scenario you follow for this example, instead, shows you how to select for monitoring event points on service components in applications already deployed and running on a server. You can see how the monitoring function fires an event whenever the processing of an application reaches one of those event points. Each of those fired events are published to the CEI server, which stores the event information about its database. You then use the Common Base Event browser to view the events.

Objectives of this tutorial

After completing this tutorial you will be able to:

- Select service component event points to monitor, with events published to the CEI server.
- View the stored events with the Common Base Event browser.

Time required to complete this tutorial

This tutorial requires approximately 15-20 minutes to complete.

Prerequisites

In order to perform this tutorial, you must have:

- Configured and started a server.
- Configured the CEI and its database.
- Enabled the diagnostic trace service on the server.
- Installed and started the Samples Gallery application on the server.
- Installed and started the business rules sample application on the server. Follow the instructions on the Samples Gallery page to set up and run the business rules sample application.

After all these prerequisites have been completed, run the business rules sample application from the Samples Gallery at least once before proceeding with the tutorial.

Example: Monitoring with the Common Event Infrastructure server:

For monitoring with the CEI server, you can use the administrative console to manage the details for event types and to display recorded events in the Common Base Event browser. This example shows the use of the console to change the level of detail recorded for some event types and to use the Common Base Event browser to view the information for individual events.

About this task

You will use the business rules sample application for this scenario; consequently, you should already have the web page containing this application already open. Keep it open, since you will be running the sample after you specify monitoring parameters. Ensure that you have already run the sample at least once, because that will cause it to appear in the list of functions that you can select to monitor.

Procedure

1. Open the administrative console.
2. In the navigation pane, click **Servers** > **Server Types** > **WebSphere application servers**.
3. Click *server_name*.
4. Under Troubleshooting, click **Logging and tracing**
5. Click **Change Log Detail levels**
6. Select the **Runtime** tab.
7. Expand the tree for **WBILocationMonitor.CEI.BR** and you will see five event types under the **WBILocationMonitor.CEI.BR.brsample.*** element. Each event type includes the name **WBILocationMonitor.CEI.BR.brsample_module.DiscountRuleGroup**, appended by the function **Operation._calculateDiscount**, and the following natures:
   - ENTRY
   - EXIT
   - FAILURE
   - SelectionKeyExtracted

- TargetFound

8. Click on each of the events and select **finest**.
9. Click **OK**.
10. Switch the business rules sample application page, and run the application once.
11. Go back to the administrative console, and select **Integration Applications** > **Common Base Event Browser** from the navigation pane.
12. If you are running your server on node within a Network Deployment environment, then you may need to modify the **`Event Data Store`** field to include to the names of your server and node. Enter the string in the following form: 'cell/nodes/*node_name*/servers/*server_name*/ejb/com/ibm/events/access/EventAccess'.
13. Press **Get Events**.

Results

You should now see a list in the upper pane of the Common Base Event browser of the four business rule events that were published to the CEI server when you ran the sample application. Select one of the events, and you will be shown the contents of the event in the lower pane. Compare this to the events published to the loggers. Notice that the browser has parsed the original XML string that was published to the CEI server, and that the business object code in the ENTRY and EXIT events was converted from the original hexadecimal format to readable XML. You may want to go back through this exercise and change the level of detail from **finest** to **fine** or **finer**, and compare the differences between the events.

After completing this exercise, you should understand how to select service component event points for monitoring using the CEI server. You have seen that the events fired in this type monitoring have a standard format, and that the results are published to a database. You should also be able to use the Common Base Event browser to retrieve events from the database, and view the information for individual events in a parsed table format on the browser.

What to do next

If you no longer want to monitor the business rules sample application, you can go back to through the steps outlined here and reset the level of detail for the sample events to **info**.

Session monitoring

You can monitor multiple events that are part of the same session, by using the Common Base Event browser to find all events on the Common Event Infrastructure database that contain the identical session ID attribute.

WebSphere Enterprise Service Bus has enhanced capabilities with which you can identify all the service component events that are part of a single session. The standard elements for the Common Base Event include an attribute under the `contextDataElement` element, called `WBISessionID`. A unique identifier for an individual session is stored in this attribute, for all service component events that were part of that session. You can use the **SessionID** field in the Common Base Event browser to search for events stored on the Common Event Infrastructure (CEI) database that match the session ID you specify. With this capability, you can easily review the process flow and contents of all the service component events.

You can use this information to assess the efficiency of your applications, and aid you in diagnosing problems that occur only under certain circumstances.

You can use the Common Base Event browser to view the returned list of events and their associated contents. If you click the All Events view, you can see columns of links for more details about events. If a particular event has a link in the **Failed** column, you can click that link to view more details about the failed event.

Related concepts:

"How do you enable monitoring?" on page 961
There are several methods that you can use to specify service component event points for monitoring, depending on the type of monitoring you are planning to do.

Chapter 11. Viewing monitored events

There are a number of ways for you to view the published results of your monitored events, depending on the type of monitoring you are using. This section presents methods that you can use to view performance data, event logs, and service component events stored on a Common Event Infrastructure database.

Viewing performance metrics with the Tivoli Performance Viewer

You can use the Tivoli Performance Viewer to start and stop performance monitoring; view Performance Monitoring Infrastructure data in chart or table form as it occurs on your system; and, optionally, log the data to a file that you can later review in the same viewer.

Before you begin

Before you can view performance metrics with the Tivoli Performance Viewer, the following conditions must be true:

- The servers that you want to monitor must be running on the node
- The Performance Monitoring Infrastructure (PMI) is enabled
- The service component event points that you want to monitor have been invoked at least once so that they can be selected from within the viewer.

About this task

The Tivoli Performance Viewer (TPV) is a powerful application that allows you view various details of about the performance of your server. The section entitled "Monitoring performance with Tivoli Performance Viewer" in the WebSphere Application Server Information Center contains details about how to use this tool for various purposes, including the resource for complete instructions on using this program. This section is limited to discussing the viewing of performance data for events specific to WebSphere Enterprise Service Bus.

The performance viewer enables administrators and programmers to monitor the current health of WebSphere ESB. Because the collection and viewing of data occurs on the process server, performance is affected. To minimize performance impacts, monitor only those servers whose activity you want to monitor.

Note: When viewing these statistics, do not mix counter-type statistics with duration-type statistics. Counters are cumulative, and the scales against which they are graphed can quickly grow depending on your application. Duration statistics, in contrast, tend to remain within a certain range because they represent the average amount of time that it takes your system to process each event. Consequently, the disparity between the statistics and their relative scales can cause one or the other type of statistic to appear skewed in the viewer graph.

Procedure

- View current performance activity
  1. Click **Monitoring and Tuning** > **Performance Viewer** > **Current Activity** in the administrative console navigation tree.

© Copyright IBM Corp. 2007, 2011

2. Select **Server**, then click the name of the server whose activity you want to monitor. You can alternatively select the check box for the server whose activity you want to monitor, then click **Start Monitoring**. To start monitoring multiple servers at the same time, select the servers then click **Start Monitoring**.
3. Select **Performance Modules**.
4. Select the check box beside the name of each performance module that you want to view. WebSphere ESB events that emit performance statistics, and that have been invoked at least once, are listed under the WBIStats.RootGroup hierarchy. Expand the tree by clicking **+** next to a node and shrink it by clicking **–** next to a node.
5. Click **View Modules**. A chart or table providing the requested data is displayed on the right side of the page. Charts are displayed by default.

   Each module has several counters associated with it. These counters are displayed in a table underneath the data chart or table. Selected counters are displayed in the chart or table. You can add or remove counters from the chart or table by selecting or clearing the check box next to them. By default, the first three counters for each module are shown.

   You can select up to 20 counters and display them in the TPV in the Current Activity mode.
6. Optional: To remove a module from a chart or table, clear the check box next to the module then click **View Modules** again.
7. Optional: To view the data in a table, click **View Table** on the counter selection table. To toggle back to a chart, click **View Graph**.
8. Optional: To view the legend for a chart, click **Show Legend**. To hide the legend, click **Hide Legend**.
9. When you have finished monitoring the performance of your events, click **Tivoli Performance Viewer**, select the server you were monitoring, and click **Stop Monitoring**.

- Log performance statistics

  While monitoring is active on a server, you can log the data from all the PMI counters that are currently enabled and record the results in a TPV log file. You can view the TPV log file for a particular time period multiple times, selecting different combinations of up to 20 counters each time. You have the flexibility to observe the relationships among different performance measures in the server during a particular period.

  1. Click **Start Logging** when viewing summary reports or performance modules.
  2. When finished, click **Stop Logging**. By default, the log files are stored in the `profile_root/logs/tpv` directory on the node on which the server is running. The TPV automatically compresses the log file when it finishes writing to it to conserve space. There must only be a single log file in each compressed file and it must have the same name as the compressed file.
  3. Click **Monitoring and Tuning** > **Performance Viewer** > **View Logs** in the administrative console navigation tree to view the logs

Viewing and interpreting service component event log files

This topic discusses how you would interpret the information in a log file generated by service component monitoring. You can view the log files in the log viewer on the administrative console, or in a separate text file editor of your choice.

Events fired to the logger by service component monitoring are encoded in Common Base Event format. When published to a log file, the event is included as a single, lengthy line of text in XML tagging format, which also includes several logger-specific fields. Consult the event catalog section of this documentation for details on deciphering the Common Base Event coding of the logged event. Use this section to understand the other fields contained in each entry of the log file, and how the format you chose for the log file when you configured the logger is structured.

Basic and advanced format fields

Logging output can be directed either to a file or to an in-memory circular buffer. If trace output is directed to the in-memory circular buffer, it must be dumped to a file before it can be viewed. Output is generated as plain text in either basic, advanced or log analyzer format as specified by the user. The basic and advanced formats for output are like the basic and advanced formats that are available for the message logs. Basic and Advanced Formats use many of the same fields and formatting techniques. The fields that can be used in these formats include:

TimeStamp
: The timestamp is formatted using the locale of the process where it is formatted. It includes a fully qualified date (YYMMDD), 24 hour time with millisecond precision and the time zone.

ThreadId
: An 8-character hexadecimal value generated from the hash code of the thread that issued the trace event.

ThreadName
: The name of the Java thread that issued the message or trace event.

ShortName
: The abbreviated name of the logging component that issued the trace event. This is typically the class name for WebSphere ESB internal components, but can be some other identifier for user applications.

LongName
: The full name of the logging component that issued the trace event. This is typically the fully qualified class name for WebSphere ESB internal components, but can be some other identifier for user applications.

EventType
: A one-character field that indicates the type of the trace event. Trace types are in lowercase. Possible values include:
  - **1** a trace entry of type fine or event.
  - **2** a trace entry of type finer.
  - **3** a trace entry of type finest, debug, or dump.
  - **Z** a placeholder to indicate that the trace type was not recognized.

ClassName
: The class that issued the message or trace event.

MethodName
: The method that issued the message or trace event.

Organization
: The organization that owns the application that issued the message or trace event.

Product
: The product that issued the message or trace event.

Component
: The component within the product that issued the message or trace event.

Basic format

Trace events displayed in basic format use the following format:

```
<timestamp><threadId><shortName><eventType>[className][methodName]<textmessage>
               [parameter 1]
               [parameter 2]
```

Advanced format

Trace events displayed in advanced format use the following format:

```
<timestamp><threadId><eventType><UOW><source=longName>[className][methodName]
<Organization><Product><Component>[thread=threadName]
<textMessage>[parameter 1=parameterValue][parameter 2=parameterValue]
```

Log analyzer format

Specifying the log analyzer format allows you to open trace output using the Log Analyzer tool, which is an application included with WebSphere Application Server. This is useful if you are trying to correlate traces from two different server processes, because it allows you to use the merge capability of the Log Analyzer.

Chapter 12. Event catalog

The event catalog contains the specifications for all the events that can be monitored for each service component type, and the associated Common Base Event extended data elements produced by each event.

Use the information presented in this section as reference material that enables you to understand how individual events are structured. This knowledge helps you decipher the information contained in each event, so that you can quickly identify the pieces of information you need from the relatively large amount of data generated by each event.

The information included in this section covers the following items:

- The structure and standard elements of the Common Base Event
- The list of WebSphere ESB-specific service components
- The extensions to the Common Base Event unique to each event type

There is also a discussion of how business objects that might be processed by a service component are captured in service component events.

When an event of a given type is fired across the Common Event Infrastructure (CEI) bus to the CEI server or to a logger, it takes the form of a Common Base Event — which is, essentially, an XML encapsulation of the event elements created according to the event catalog specification. The Common Base Event includes a set of standard elements, server component identification elements, Event Correlation Sphere identifiers, and additional elements unique to each event type. All of these elements are passed to the CEI server or logger whenever an event is fired by a service component monitor, with one exception: if the event includes the business object code within the payload, you may specify the amount of business object data that you want to include in event.

The Common Base Event standard elements

The elements of the Common Base Event that are included in all events fired from service component monitoring are listed here.

Attribute	Description
version	Set to `1.0.1`.
creationTime	The time at which the event is created, in UTC.
globalInstanceId	The identifier of the Common Base Event instance. This ID is automatically generated.
localInstanceId	This ID is automatically generated (might be blank).
severity	The impact that the event has on business processes . This attribute is set to `10` (information). Otherwise, it is not used.
priority	Not used.
reporterComponentId	Not used.
locationType	Set to `Hostname`.
location	Set to the host name of the executing server.
application	Not used.

© Copyright IBM Corp. 2007, 2011

Attribute	Description
executionEnvironment	A string that identifies the operating system.
componentType	The component QName, based on the Apache QName format. For business processes, set to: `www.ibm.com/namespaces/autonomic/Workflow_Engine`
subComponent	The observable element name. For business processes, set to `BFM`. .
componentIdType	Set to `ProductName`.
instanceId	The identifier of the server. This identifier has the format *`cell_name/node_name/server_name`*. The delimiters are operating system dependent.
processId	The process identifier of the operating system.
threadId	The thread identifier of the Java virtual machine (JVM).
Situation Type	The type of situation that caused the event to be reported. For WebSphere ESB specific components, set to `ReportSituation`.
Situation Category	The category of the type of situation that caused the event to be reported. For WebSphere ESB specific components, set to `STATUS`.
Situation Reasoning Scope	The scope of the impact of the situation reported. For WebSphere ESB specific components, set to `EXTERNAL`.
ECSCurrentID	The value of the current Event Correlation Sphere ID.
ECSParentID	The value of the parent Event Correlation Sphere ID.
WBISessionID	The value of the current Session ID.
extensionName	Set to the event name.

Business objects in events

Business object data is carried within the event in XML format. The Common Base Event format includes an `xs:any` schema, which encapsulates the business object payload in XML elements.

You specify the level of business object detail that will be captured in service component events. This level of detail affects only the amount of business object code that will be passed to the event; all the other Common Base Event elements (both standard and event-specific) will be published to the event.

The names of the detail levels applicable to service component events differ depending on whether you created a static monitor using IBM Integration Designer or a dynamic monitor on the administrative console, but they correspond as shown in the following table:

Administrative console detail level	Common Base Event/Integration Designer detail level	Payload information published
FINE	EMPTY	None.
FINER	DIGEST	Payload description only.
FINEST	FULL	All of the payload.

The detail level is specified by the `PayloadType` element, which is part of the event instance data. The actual business object data is included in the event only if the monitor is set to record `FULL/FINEST` detail.

The business object data itself is included in the Common Base Event under an `xsd:any` schema. You can see the process server business object payloads with the root element named `wbi:event`.

If you are publishing the event output to the logger, you will see the output when you view the log files. If the event is published to the CEI server, you can use the Common Base Event browser to view the event. You can then click the `wbi:event` link to view the business object data.

WebSphere Enterprise Service Bus events

WebSphere ESB features its own service components, and each of these components has its own set of event points that can be monitored.

Service components contain one or more elements, which are sets of different steps processed in each service component. In turn, each element has its own set of event natures, that are key points that are reached when processing a service component element. All service components, their elements and associated event natures, and the extended data elements unique to each event are listed.

Resource Adapter events

The event types available for the resource adapter component are listed.

The elements of the resource adapter component (base name `eis:WBI.JCAAdapter`) that can be monitored are listed here, along with their associated event natures, event names, and the extended data elements that are unique to each event.

Event Name	Event Natures	Event Contents	Type
`InboundEventRetrieval` element			
eis:WBI.JCAAdapter. InboundEventRetrieval. ENTRY	ENTRY	pollQuantity	int
		status	int
		eventTypeFilters	string
eis:WBI.JCAAdapter. InboundEventRetrieval. EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. InboundEventRetrieval. FAILURE	FAILURE	FailureReason	exception
`InboundEventDelivery` element			
eis:WBI.JCAAdapter. InboundEventDelivery.ENTRY	ENTRY	N/A	
eis:WBI.JCAAdapter. InboundEventDelivery.EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. InboundEventDelivery.FAILURE	FAILURE	FailureReason	exception
`Outbound` element			
eis:WBI.JCAAdapter. Outbound.ENTRY	ENTRY	N/A	
eis:WBI.JCAAdapter. Outbound.EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. Outbound.FAILURE	FAILURE	FailureReason	exception

Event Name	Event Natures	Event Contents	Type
InboundCallbackAsyncDeliverEvent element			
eis:WBI.JCAAdapter. InboundCallbackAsyncDeliverEvent. ENTRY	ENTRY	N/A	
eis:WBI.JCAAdapter. InboundCallbackAsyncDeliverEvent. EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. InboundCallbackAsyncDeliverEvent. FAILURE	FAILURE	FailureReason	exception
InboundCallbackSyncDeliverEvent element			
eis:WBI.JCAAdapter. InboundCallbackSyncDeliverEvent. ENTRY	ENTRY	N/A	
eis:WBI.JCAAdapter. InboundCallbackSyncDeliverEvent. EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. InboundCallbackSyncDeliverEvent. FAILURE	FAILURE	FailureReason	exception
Polling element			
eis:WBI.JCAAdapter. Polling.STARTED	STARTED	PollFrequency	int
		PollQuantity	int
eis:WBI.JCAAdapter. Polling.STOPPED	STOPPED	N/A	
Delivery element			
eis:WBI.JCAAdapter. Delivery.EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. Delivery.FAILURE	FAILURE	EventID	string
		FailureReason	exception
Retrieval element			
eis:WBI.JCAAdapter. Retrieval.FAILURE	FAILURE	EventID	string
		FailureReason	exception
Endpoint element			
eis:WBI.JCAAdapter. Endpoint.FAILURE	FAILURE	FailureReason	exception
Recovery element			
eis:WBI.JCAAdapter. Recovery.EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. Recovery.FAILURE	FAILURE	FailureReason	exception
EventFailure element			
eis:WBI.JCAAdapter. EventFailure.FAILURE	FAILURE	FailureReason	exception
Connection element			
eis:WBI.JCAAdapter. Connection.FAILURE	FAILURE	FailureReason	exception

Binding events

The event types available for the WebSphere ESB bindings (JMS, JAX-WS, HTTP, EJB, and EIS) are listed.

All binding events contain a single element, with a base prefix name of binding:WBI.SCA.*. The event structure is described in the XML Schema Definition (XSD) file `SCABindingEvents.xsd`. The file can be found in the `$WAS_HOME/plugins/com.ibm.ws.sca.bindingcore.jar/model` directory.

Note: To enable the monitor event framework, open the following trace:

```
WBIEventMonitor.CEI.SCABinding.*=all: WBIEventMonitor.LOG.SCABinding.*=all
```

Events for the JMS, JAX-WS, HTTP, EJB, and EIS bindings are listed in the sections that follow.

JMS binding

Table 104. JMS binding

Event Name	Event Nature	Event Contents	Type
WBI.SCA.JMSBINDING.ENTRY	ENTRY	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: JMS
		DESTINATION	string
		REPLY_DESTINATION	string
		CALLBACK_DESTINATION	string
		DIRECTION	string restriction: REQUEST, RESPONSE
		JMS_TYPE	string restriction: SIBus, GenericJMS, MQJMS, MQNative
		DATABINDING_NAME	string
		MESSAGE_ID	string
		CORRELATION_ID	string

Table 104. JMS binding (continued)

Event Name	Event Nature	Event Contents	Type
WBI.SCA.JMSBINDING.EXIT	EXIT	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: JMS
		DESTINATION	string
		REPLY_DESTINATION	string
		CALLBACK_DESTINATION	string
		DIRECTION	string restriction: REQUEST, RESPONSE
		JMS_TYPE	string restriction: SIBus, GenericJMS, MQJMS, MQNative
		DATABINDING_NAME	string
		MESSAGE_ID	string
		CORRELATION_ID	string
WBI.SCA.JMSBINDING.FAILURE	FAILURE	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: JMS
		DESTINATION	string
		REPLY_DESTINATION	string
		CALLBACK_DESTINATION	string
		DIRECTION	string restriction: REQUEST, RESPONSE
		JMS_TYPE	string restriction: SIBus, GenericJMS, MQJMS, MQNative
		DATABINDING_NAME	string
		MESSAGE_ID	string
		CORRELATION_ID	string
		Exception	string

For an example of JMS binding event code, see the related links at the end of this topic.

JAX-WS binding

Events associated with the JAX-WS binding, along with the extended data elements that are unique to each event, are listed in Table 105.

Table 105. JAX-WS binding

Event Name	Event Nature	Event Contents	Type
WBI.SCA.JAXWSBINDING.ENTRY	ENTRY	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: JAX-WS
		SERVICE_QNAME	string
		PORT_QNAME	string
		ENDPOINT	string
WBI.SCA.JAXWSBINDING.EXIT	EXIT	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: JAX-WS
		SERVICE_QNAME	string
		PORT_QNAME	string
		ENDPOINT	string
WBI.SCA.JAXWSBINDING.FAILURE	FAILURE	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: JAX-WS
		SERVICE_QNAME	string
		PORT_QNAME	string
		ENDPOINT	string
		Exception	string

HTTP binding

Events associated with the HTTP binding, along with the extended data elements that are unique to each event, are listed in Table 106 on page 1008.

Table 106. HTTP binding events

Event Name	Event Nature	Event Contents	Type
WBI.SCA.HTTPBINDING.ENTRY	ENTRY	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: HTTP
		ENDPOINT	string
		HTTPMETHOD	string
WBI.SCA.HTTPBINDING.EXIT	EXIT	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: HTTP
		ENDPOINT	string
		HTTPMETHOD	string
WBI.SCA.HTTPBINDING.FAILURE	FAILURE	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: HTTP
		ENDPOINT	string
		HTTPMETHOD	string
		Exception	string

EJB binding

Events associated with the EJB binding, along with the extended data elements that are unique to each event, are listed in Table 107 on page 1009.

Table 107. EJB binding events

Event Name	Event Nature	Event Contents	Type
WBI.SCA.EJBBINDING.ENTRY	ENTRY	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: EJB
		JNDI_NAME	string
		DATABINDING_NAME	string
WBI.SCA.EJBBINDING.EXIT	EXIT	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: EJB
		JNDI_NAME	string
		DATABINDING_NAME	string
WBI.SCA.EJBBINDING.FAILURE	FAILURE	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: EJB
		JNDI_NAME	string
		DATABINDING_NAME	string
		Exception	string

EIS binding

Events associated with the EIS binding, along with the extended data elements that are unique to each event, are listed in Table 108.

Table 108. EIS binding events

Event Name	Event Nature	Event Contents	Type
WBI.SCA.EISBINDING.ENTRY	ENTRY	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: EIS
		RESOURCE_ADAPTER	string

Table 108. EIS binding events (continued)

Event Name	Event Nature	Event Contents	Type
WBI.SCA.EISBINDING.EXIT	EXIT	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: EIS
		RESOURCE_ADAPTER	string
WBI.SCA.EISBINDING.FAILURE	FAILURE	MODULE_NAME	string
		BINDING_NAME	string
		OPERATION_NAME	string
		BINDING_TYPE	string restriction: import and export
		BINDING_PROTOCOL	string restriction: EIS
		JNDI_NAME	string
		RESOURCE_ADAPTER	string
		Exception	string

Generic JMS binding event code example

Here is a sample of the code for a generic JMS binding event.

Generic JMS binding event

```
<wbi:event
xmlns:wbi="http://www.ibm.com/xmlns/prod/websphere/monitoring/6.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:binding="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0:Binding">
<wbi:eventHeaderData>
<wbi:WBISESSION_ID>9.115.71.147;TestGenJMSClt;Import1;operation3;1311156316324;520298243</wbi:WBISESSION_ID>
<wbi:ECSCurrentID>9.115.71.147;TestGenJMSClt;TestIntfPartner;Import1;operation3;1311156316324;520298243
</wbi:ECSCurrentID>
<wbi:ECSParentID>9.115.71.147;TestGenJMSClt;Import1;operation3;1311156316324;520298243</wbi:ECSParentID>
<wbi:WBIEventVersion>6.1</wbi:WBIEventVersion>
</wbi:eventHeaderData>
<wbi:eventPointData xsi:type="binding:WBI.SCA.JMSBINDING.EXIT">
<wbi:eventNature>EXIT</wbi:eventNature>
<wbi:payloadType>bogus</wbi:payloadType>
<binding:MODULE_NAME>TestGenJMSClt</binding:MODULE_NAME>
<binding:BINDING_NAME>Import1</binding:BINDING_NAME>
<binding:OPERATION_NAME>operation3</binding:OPERATION_NAME>
<binding:BINDING_TYPE>Import</binding:BINDING_TYPE>
<binding:BINDING_PROTOCOL>JMS</binding:BINDING_PROTOCOL>
<binding:DESTINATION>queue:///BDRQ?version=6</binding:DESTINATION>
<binding:REPLY_DESTINATION>queue:///BDSQ?version=6</binding:REPLY_DESTINATION>
<binding:CALLBACK_DESTINATION>queue://TestGenJMSClt.Import1_GENJMS_CALLBACK_D_SIB?busName
=SCA.SYSTEM.WIN-1OPBE8FWO28Node01Cell.Bus</binding:CALLBACK_DESTINATION>
<binding:DIRECTION>REQUEST</binding:DIRECTION>
<binding:JMS_TYPE>GenericJMS</binding:JMS_TYPE>
<binding:DATABINDING_NAME>com.ibm.wsspi.sca.jms.data.JMSDataHandlerBindingImpl</binding:DATABINDING_NAME>
<binding:MESSAGE_ID>ID:414d5120513238382020202020202020637c264e20034307</binding:MESSAGE_ID>
<binding:CORRELATION_ID>ID:414d5120513238382020202020202020637c264e20034307</binding:CORRELATION_ID>
</wbi:eventPointData>
</wbi:event>
```

Business rule events

The event types available for the business rule component are listed.

The business rule component (base name `br:WBI.BR`) contains a single element that can be monitored. All event types for this element are listed here, with their associated event natures, event names, and the extended data elements that are unique to each event.

Event Name	Event Nature	Event Contents	Type
br:WBI.BR.ENTRY	ENTRY	operationName	string
br:WBI.BR.EXIT	EXIT	operationName	string
br:WBI.BR.FAILURE	FAILURE	ErrorReport	Exception
		operationName	string
WBI.BR. br:SelectionKeyExtracted	SelectionKeyExtracted	operationName	string
br:WBI.BR.TargetFound	TargetFound	operationName	string
		target	string

Business state machine events

The event types available for the business state machine component are listed.

The elements from the business state machine component (base name `bsm:WBI.BSM`) that can be monitored are listed here, along with their associated event natures, event names, and all extended data elements that are unique to each event.

Event Name	Event Nature	Event Contents	Type
`StateMachineDefinition` element			
bsm:WBI.BSM. StateMachineDefinition. ALLOCATED	ALLOCATED	instanceID	string
bsm:WBI.BSM. StateMachineDefinition. RELEASED	RELEASED	instanceID	string
`Transition` element			
bsm:WBI.BSM.Transition.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.Transition.EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.Transition.FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
`State` element			
bsm:WBI.BSM.State.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.State.EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.State.FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
`Guard` element			

Event Name	Event Nature	Event Contents	Type
bsm:WBI.BSM.Guard.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.Guard.EXIT	EXIT	instanceID	string
		name	string
		result	boolean
bsm:WBI.BSM.Guard.FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
`Action` element			
bsm:WBI.BSM.Action.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.Action.EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.Action.FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
`EntryAction` element			
bsm:WBI.BSM.EntryAction. ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.EntryAction. EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.EntryAction. FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
`ExitAction` element			
bsm:WBI.BSM.ExitAction.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.ExitAction.EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.ExitAction. FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
`Timer` element			
bsm:WBI.BSM.Timer.START	START	instanceID	string
		name	string
		duration	string
bsm:WBI.BSM.Timer.STOPPED	STOPPED	instanceID	string
		name	string
		duration	string

Map events

The event types available for the map component are listed.

The elements from the map component (base name `map:WBI.MAP`) that can be monitored are listed here, along with their event natures, event names, and all extended data elements that are unique to each event.

Table 109. Base element

Event Name	Event Nature	Event Contents	Type
map:WBI.MAP.ENTRY	ENTRY	N/A	N/A
map:WBI.MAP.EXIT	EXIT	N/A	N/A
map:WBI.MAP.FAILURE	FAILURE	FailureReason	Exception
`Transformation` element			
map:WBI.MAP.Transformation. ENTRY	ENTRY	N/A	N/A
map:WBI.MAP.Transformation. EXIT	EXIT	N/A	N/A
map:WBI.MAP.Transformation. FAILURE	FAILURE	FailureReason	Exception

Mediation events

The event types available for the mediation component are listed.

The elements from the mediation component (base name `ifm:WBI.MEDIATION`) that can be monitored are listed here, along with their associated event natures, names, and all extended data elements that are unique to each event.

Event Name	Event Nature	Event Contents	Type
`OperationBinding` element			
ifm:WBI.MEDIATION. OperationBinding.ENTRY	ENTRY	InteractionType	string
		TicketID	string
		Source	string
		Target	string
ifm:WBI.MEDIATION. OperationBinding.EXIT	EXIT	InteractionType	string
		TicketID	string
		Source	string
		Target	string
ifm:WBI.MEDIATION. OperationBinding.FAILURE	FAILURE	InteractionType	string
		TicketID	string
		Source	string
		Target	string
		ErrorReport	Exception
`ParameterMediation` element			
ifm:WBI.MEDIATION. ParameterMediation. ENTRY	ENTRY	Type	string
		TransformName	string
WBI.MEDIATION. ParameterMediation. EXIT	EXIT	Type	string
		TransformName	string

Event Name	Event Nature	Event Contents	Type
ifm:WBI.MEDIATION. ParameterMediation. FAILURE	FAILURE	Type	string
		TransformName	string
		ErrorReport	Exception

Recovery events

The event types available for the recovery component are listed.

The recovery component (base name `recovery:WBI.Recovery`) contains a single element that can be monitored. All event types for this element are listed here, along with their associated event natures, event names, and the extended data elements that are unique to each event.

Event Name	Event Nature	Event Contents	Type
recovery:WBI.Recovery. FAILURE	FAILURE	MsgId	string
		DestModuleName	string
		DestComponentName	string
		DestMethodName	string
		SourceModuleName	string
		SourceComponentName	string
		ResubmitDestination	string
		ExceptionDetails	string
		SessionId	string
		FailureTime	dateTime
		ExpirationTime	dateTime
		Status	int
		MessageBody	byteArray
		Deliverable	boolean
recovery:WBI.Recovery. DEADLOOP	DEADLOOP	DeadloopMsgId	string
		SIBusName	string
		QueueName	string
		Reason	string
recovery:WBI.Recovery. RESUBMIT	RESUBMIT	MsgId	string
		OriginalMesId	string
		ResubmitCount	int
		Description	string
recovery:WBI.Recovery. DELETE	DELETE	MsgId	string
		deleteTime	dateTime
		Description	string

Service Component Architecture events

The event types available for the Service Component Architecture are listed.

The Service Component Architecture (SCA) contains a single element, with a base name of `sca:WBI.SCA.MethodInvocation`. All the events and associated natures of this element are listed here, along with all extended data elements and that are unique to each event.

Note: Do not confuse these events with SCA-specific Application Response Measurement (ARM) performance statistics.

Event Name	Event Nature	Event Contents	Type
WBI.SCA. MethodInvocation. ENTRY	ENTRY	SOURCE COMPONENT	string
		SOURCE INTERFACE	string
		SOURCE METHOD	string
		SOURCE MODULE	string
		SOURCE REFERENCE	string
		TARGET COMPONENT	string
		TARGET INTERFACE	string
		TARGET METHOD	string
		TARGET MODULE	string
WBI.SCA. MethodInvocation. EXIT	EXIT	SOURCE COMPONENT	string
		SOURCE INTERFACE	string
		SOURCE METHOD	string
		SOURCE MODULE	string
		SOURCE REFERENCE	string
		TARGET COMPONENT	string
		TARGET INTERFACE	string
		TARGET METHOD	string
		TARGET MODULE	string
WBI.SCA. MethodInvocation. FAILURE	FAILURE	SOURCE COMPONENT	string
		SOURCE INTERFACE	string
		SOURCE METHOD	string
		SOURCE MODULE	string
		SOURCE REFERENCE	string
		TARGET COMPONENT	string
		TARGET INTERFACE	string
		TARGET METHOD	string
		TARGET MODULE	string
		Exception	string

Selector events

The event types available for the Selector component are listed.

The selector component contains a single element that can be monitored. All event types for this element are listed here, along with their associated event natures, event names, and the extended data elements that are unique to each event. All selector events have a base name of `sel:WBI.SEL`.

Event Name	Event Nature	Event Contents	Type
sel:WBI.SEL.ENTRY	ENTRY	operationName	string
sel:WBI.SEL.EXIT	EXIT	operationName	string
sel:WBI.SEL.FAILURE	FAILURE	ErrorReport	Exception
		operationName	string
sel:WBI.SEL. SelectionKeyExtracted	SelectionKeyExtracted	operationName	string
sel:WBI.SEL.TargetFound	TargetFound	operationName	string
		target	string

Chapter 13. Samples

Samples installed from either the product or from the Web are based on real business problems to demonstrate the capabilities of WebSphere ESB.

Samples included with the product

There is currently one WebSphere ESB sample available from the Samples Gallery, which you can install with the product. This sample demonstrates services being integrated to provide new features for customers. The sample includes:

- 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 Gallery is available to install during WebSphere ESB installation and is selected for installation by default. The sample is installed in the *install_root*/samples directory.

To access the WebSphere ESB samples:

1. Start the server.
2. Start the Samples Gallery by selecting Samples Gallery on the First Steps panel.
3. The WebSphere ESB samples are listed as installed samples in the Samples Gallery.
4. From the navigation menu on the left, click **Installed Samples** > **Business Integration** and click on a sample.

Samples available on the Web

The product has a samples download site, described as the online Samples Gallery, that lists additional samples. New WebSphere ESB samples are added here periodically. You can access the samples Web site at http://publib.boulder.ibm.com/bpcsamp/index.html.

Note: The samples are for demonstration purposes only and might be updated in subsequent releases. The code provided is not intended to run in a secured production environment.

© IBM Corporation 2005, 2011 © IBM 2007, 2011

Related information:

PDF documentation
WebSphere Enterprise Service Bus documentation (in PDF format)

Information roadmaps
Business Process Management information roadmaps on IBM developerWorks organize information about WebSphere ESB and the other products in the portfolio.

IBM Education Assistant
Multimedia educational modules about WebSphere ESB, provided by IBM Education Assistant.

Technotes
WebSphere ESB Support > technote search for 7.5 samples category documents.

Overview
Overview tab, on product library web page. Use this page to access announcements, data sheets, and other general library documents related to WebSphere ESB.

Running the StockQuote sample

The WebSphere ESB StockQuote sample demonstrates how you can easily integrate disparate services and manipulate in-flight messages without affecting the applications. This is achieved through the Service Component Architecture (SCA) and mediation functionality provided in WebSphere ESB.

About this task

A financial services company provides an interactive web-based stock market service to its customers. The company wants to differentiate itself from its competition by offering tiered levels of service. They want to offer delayed stock quotes to their *standard* customers and real-time quotes to their *premium* customers (customers who pay a subscription).

The company wants to:

- Make the delayed stock quote service available to all customers.
- Log all requests to the service to satisfy audit requirements.
- Make the real-time stock quote service available to *premium* customers.

Figure 316. StockQuote overview

To run the StockQuote sample:

Procedure

1. Launch the First Steps console.
2. Start the server if it is not already started.
3. Start the Samples Gallery by clicking **Samples Gallery** on the First Steps panel. The WebSphere ESB StockQuote sample is listed as an installed sample in the Samples Gallery.
4. Click **Installed samples** > **Business Integration** > **StockQuote**.
5. Check the address of the Samples Gallery in the address field of the browser window. If the port number value is 9080, go to the next step, otherwise perform the following substeps.
   a. Open the administrative console and log in.
   b. Click **Applications** > **Enterprise Applications**. Available applications are displayed.
   c. Click **StockQuote**. The details of the StockQuote application are displayed.
   d. In the Module section, click **Manage Modules**.
   e. Click **StockQuoteEJB**. The details of the StockQuoteEJB.jar file are displayed.
   f. In the Web Services Properties section, click **Web services client bindings**.
   g. For `sca/import/RealtimeService`, click **Edit** in the Port Information column. The port information is displayed.
   h. In the Overridden Endpoint URL field, type `http://hostname:`*`nnnn`*`/RealtimeService/services/RealtimeServiceSOAP`, where *hostname* is either

localhost or the host name of the workstation where the server is running; and*nnnn* is the port number used for your default host.

i. Click **OK**.

j. For sca/import/DelayedService, click **Edit** in the Port Information column. The port information is displayed.

k. In the Overridden Endpoint URL field, type http://hostname:*nnnn*/DelayedService/services/DelayedServiceSOAP, where *hostname* is either localhost or the host name of the workstation where the server is running; and*nnnn* is the port number used for your default host.

l. Click **OK**.

m. Save your workspace changes to the master configuration and return to the Samples Gallery.

The sample can now be run successfully on your server.

6. Click **Run** and perform the following actions with each of the 2 customers to demonstrate the capabilities of the integrated service:

   a. Select a customer from the drop-down list.

   b. Select one or more symbols.

   c. Click the Request button.

Results

Customer A, *a standard customer*, will receive time delayed quotes. Customer B, *a premium customer*, will receive real-time quotes.

What to do next

You can learn about the artifacts used in the sample from the Technotes provided in the Samples Gallery.

Uninstalling the StockQuote sample

You can uninstall the StockQuote sample using a script which you run from the command line. This allows you to remove the sample from your profile.

About this task

If you choose to install the samples gallery when you install WebSphere ESB, the StockQuote sample is also installed. This means that every stand-alone profile you subsequently create will have the sample installed. If you want to try out the sample on a stand-alone profile and then have a clean profile to start working with, you can uninstall the sample using this method.

Procedure

1. Start the server.

2. From the command line, type the following command on a Windows platform.

```
wsadmin -f install_root\samples\bin\ESBStockQuoteSample\uninstall.jacl
-cell cell_name -node node_name -dataSourceName CustomerDatabase
-connectionFactoryName StockQuoteReplyQCF -queueName StockQuoteService_Export
-activationSpecName StockQuoteService_ExportAS
-destinationName StockQuoteService_Export
```

   or for all other platforms, type the following command:

```
wsadmin -f install_root/samples/bin/ESBStockQuoteSample/uninstall.jacl
-cell cell_name -node node_name -dataSourceName CustomerDatabase
-connectionFactoryName StockQuoteReplyQCF -queueName StockQuoteService_Export
-activationSpecName StockQuoteService_ExportAS
-destinationName StockQuoteService_Export
```

where *install_root* represents the location of yourWebSphere ESB installation, *cell_name* and *node_name* represent the cell and node names for your profile.

Results

This script will remove the StockQuote sample ear files from the server and remove all the resources created for the sample.

List of samples in the online Samples Gallery

The online Samples Gallery contains samples based on real business problems to demonstrate the capabilities of WebSphere ESB.

You can access the online Samples Gallery at http://publib.boulder.ibm.com/bpcsamp/index.html.

The iteration aggregation sample

The WebSphere ESB aggregation sample demonstrates the use of iteration to achieve aggregation of data.

About this task

The iteration aggregation sample is based on a user submitting a batch of orders within a single request. Each order contains a customer identifier, an item identifier and a quantity. Time can be saved by submitting multiple orders in a single batch, identified by an identifier. The system will handle the batch and return the status information back to the user.

Before the batch can be submitted to the dispatch backend service, each order in the batch must be enriched with customer mailing information from a customer service. What is not shown within the Procedure section is that the response flow, which translates the status information returned by the dispatch system to the store response format.

Procedure

1. Store the batch identifier.
2. For each order, build a message to send to the Customer service. If there are no orders in the batch, to save unnecessary calls to the dispatch backend service, build and return an empty response.
3. Call the Customer service to retrieve mailing information for the customer. If there are any errors, stop processing.
4. Store the result in the shared aggregation context.
5. Add the result to the list of orders already enriched.
6. Return to step 2 to process the next order.
7. When all orders are completed, build the message to send to the dispatch system, including the newly enriched order batch.
8. Send the new message to the dispatch system.

Results

In the iteration aggregation sample, the customer and dispatch services are implemented as Java objects exposed as SOAP/HTTP web services, while the mediation itself is exposed as a SOAP/HTTP web service. Shared interfaces and business objects are held in a separate library called StoreLib. The whole system can be tested by using the included StoreClient web application which allows a user to build a batch, submit it to the mediation, and view the returned results.

Related information:

This sample in the Samples Gallery

The MQ data binding sample

The WebSphere ESB MQ data binding sample demonstrates how to write a MQ data binding.

About this task

All artifacts required to run the MQ data binding sample, are located in the following directories:

- *install_root*/samples/bin/ESBMQBinding for the jar file and jython script.
- *install_root*/samples/lib/ESBMQBinding for the ear file.

Procedure

1. Use the MQ explorer to set up the MQ Server. You will need to have a queue manager started, with four queues:
   - Request input queue
   - Request output queue
   - Response input queue
   - Response output queue

   The MQ server will also require a running MQ listener.
2. Run the Jython script MQDataBindingJNDIResources.py to create the JNDI resources needed by the sample. This will create the import and export connection factories, and the four queues. The syntax for running the script is:

```
Wsadmin –lang jython –f MQDataBindingJNDIResources.py [–mqhost hostname] [–mqport portnumber] –qmname qmname
–request_input_q req_in_q –request_output_q
req_out_q –response_input_q resp_in_q –response_output_q
resp_out_q [-clean]
```

3. For the data binding test, deploy the MQ data binding sample EAR file, named MQDataBindingMediation.ear to the WebSphere ESB server and start the application.
4. Restart the server to connect to the MQ server.
5. On the MQ server machine, run the test program to put a test message on to the request input queue. Use the following command:

```
java MQTest queueManager request_input_Queue request_output_Queue response_input_Queue response_ouptut_Queue
```

   The MQ server machine will put a delimited message on to the request input queue, with a “,” as the delimiter. The MQ data binding sample will pick up this message, change the header, and output a fixed length message on to the request output queue. The message has a fixed length of 15 characters per element, and a ‘~' as the padding character. The MQTest class will then pick up the message and display the header and message content, as the example is shown below:

```
Sending message: 23,John,Doe,address,10,577 Airport Blvd,Burlingame,CA,94010,
to the req_in queue.

Message received from the req_out queue:
strucID     = WESB
strucLength = 20
msgType     = FXLN
msgOption1  = 15
msgOption2  = ~
Message length is 135
Message type is 1
Received message: 23~~~~~~~~~~~~~~~John~~~~~~~~~~~~Doe~~~~~~~~~~~~~~address~~~~~~~~~10~~~~~~~~~~~~~~~577 Airport BlvBurlingame
~~~~~CA~~~~~~~~~~~~~~94010~~~~~~~~~~
```

The message is then forwarded to the response input queue. The response mediation flow will pick up the message, change it back to a delimited message, and output it onto the response output queue. MQTest will pick up the final message from the response output queue and display the header and message, as the example is shown below.

```
Sending message: 23~~~~~~~~~~~~~~~John~~~~~~~~~~~~Doe~~~~~~~~~~~~~~address~~~~~~~~~10~~~~~~~~~~~~~~~577 Airport BlvBurlingame
~~~~~CA~~~~~~~~~~~~~~94010~~~~~~~~~~
to the resp_in queue.

Message received from the resp_out queue:
strucID     = WESB
strucLength = 20
msgType     = DELI
msgOption1  = ,
msgOption2  =
Message length is 59
Message type is 2
Received message: 23,John,Doe,address,10,577 Airport Blv,Burlingame,CA,94010,
```

Related information:

This sample in the Samples Gallery

The XSLT mapping file migration sample

The XSLT mapping file migration sample demonstrates how you can migrate XSLT mapping files that have been created in Integration Designer Version 6.0.2 to the new format used in Version 6.2 or later.

About this task

XSLT mapping files created with Integration Designer Version 6.0.2 uses the XSL transform mapping editor, that has been replaced with an enhanced mapping editor in later versions.

The sample provides a Project Interchange file called `stockquotesample.zip` which was created using Integration Designer Version 6.0.2. The mediation flow inside contains the four XSL transform primitives that need migrating.

This sample is imported, migrated and built using Integration Designer [WID].

Instructions for downloading, building, and running this sample are provided in the Samples Gallery **Mediation features** > **XSLT mapping file migration**.

Related information:

This sample in the Samples Gallery

The service gateway sample

The WebSphere Enterprise Service Bus (WebSphere ESB) service gateway sample extends the capabilities of the Static and Dynamic Service Gateway patterns that were introduced in WebSphere ESB V6.2. This sample creates a proxy gateway, which is a type of service gateway.

About this task

A proxy gateway is based on the goal of a service gateway: to provide a generic capability across a number of service providers. Use the Proxy Gateway wizard to build a proxy gateway:

Procedure

1. Open IBM Integration Designer in a new workspace.
2. Select **Window** > **Show view** > **Patterns Explorer** to open the Patterns view.
3. Create a new Proxy Gateway in the Patterns Explorer tab.
   a. Right-click **Proxy Service Gateway** and select **Create New Instance**.
   b. Enter `MyProxyGateway` as the Integration Solution name and click **OK**.

   The New Pattern Instance panel opens
4. Accept the default configuration for the Proxy Gateway and click **Generate**. The Proxy Gateway wizard generates the required artifacts and a summary panel opens.
5. Scroll down and click the **requestResponse Request Flow** diagram to open the Mediation Flow Editor. A single Gateway Endpoint Lookup primitive is created by default.
6. Select the **Gateway Endpoint Lookup Primitive**.

Results

The Proxy Gateway has now been created and the mediation module can be deployed in the runtime environment.

Related information:

This sample in the Samples Gallery

The Endpoint Lookup sample

The WebSphere Enterprise Service Bus endpoint lookup sample can retrieve a list of endpoints for a service from WSRR. The endpoints can be passed as a list to a callout which tries each endpoint in turn until one is successful.

Before you begin

Download a WebSphere Integration Developer project called `EndpointLookupSample.zip` from the Download page of the sample on the Samples Gallery.

About this task

The sample also shows combining retry capability in a mediation flow callout with a list of service endpoints to provide failover capability. The sample shows what happens when one of the OrderDispatchService services becomes unavailable. At this point the mediation callout will fail over to another OrderDispatchService service endpoint in the list. To build the Endpoint Lookup sample, complete these steps which are further described in the sample:

Procedure

1. Create and edit a mediation module.
2. Add a WebService Export.
3. Build the mediation flow.

Results

The mediation module can now be tested and then deployed in the runtime environment.

Related information:

This sample in the Samples Gallery

Chapter 14. Overview of troubleshooting

Troubleshooting is a systematic approach to solving a problem. The goal is to determine why something does not work as expected and how to resolve the problem.

The first step in the troubleshooting process is to describe the problem completely. Without a problem description, neither you or IBM can know where to start to find the cause of the problem. This step includes asking yourself basic questions, such as:

- What are the symptoms of the problem?
- Where does the problem occur?
- When does the problem occur?
- Under which conditions does the problem occur?
- Can the problem be reproduced?

The answers to these questions typically lead to a good description of the problem, and that is the best way to start down the path of problem resolution.

What are the symptoms of the problem?

When starting to describe a problem, the most obvious question is "What is the problem?" Which might seem like a straightforward question; however, you can break it down into several more-focused questions that create a more descriptive picture of the problem. These questions can include:

- Who, or what, is reporting the problem?
- What are the error codes and messages?
- How does the system fail? For example, is it a loop, hang, lock up, performance degradation, or incorrect result?
- What is the business impact of the problem?

Where does the problem occur?

Determining where the problem originates is not always simple, but it is one of the most important steps in resolving a problem. Many layers of technology can exist between the reporting and failing components. Networks, disks, and drivers are only a few components to be considered when you are investigating problems.

The following questions can help you to focus on where the problem occurs in order to isolate the problem layer.

- Is the problem specific to one platform or operating system, or is it common for multiple platforms or operating systems?
- Is the current environment and configuration supported?

Remember that if one layer reports the problem, the problem does not necessarily originate in that layer. Part of identifying where a problem originates is understanding the environment in which it exists. Take some time to completely describe the problem environment, including the operating system and version, all corresponding software and versions, and hardware information. Confirm that you are running within an environment that is a supported configuration; many

© Copyright IBM Corp. 2007, 2011

problems can be traced back to incompatible levels of software that are not intended to run together or have not been fully tested together.

When does the problem occur?

Develop a detailed timeline of events leading up to a failure, especially for those cases that are one-time occurrences. You can most simply do this by working backward: Start at the time an error was reported (as precisely as possible, even down to the millisecond), and work backward through the available logs and information. Typically, you need to look only as far as the first suspicious event that you find in a diagnostic log; however, this is not always simple to do and takes practice. Knowing when to stop looking is especially difficult when multiple layers of technology are involved, and when each has its own diagnostic information.

To develop a detailed timeline of events, answer the following questions:

- Does the problem happen only at a certain time of day or night?
- How often does the problem happen?
- What sequence of events leads up to the time that the problem is reported?
- Does the problem happen after an environment change, such as upgrading or installing software or hardware?

Responding to these types of questions can provide you with a frame of reference in which to investigate the problem.

Under which conditions does the problem occur?

Knowing what other systems and applications are running at the time that a problem occurs is an important part of troubleshooting. These and other questions about your environment can help you to identify the root cause of the problem:

- Does the problem always occur when the same task is being performed?
- Does a certain sequence of events need to occur for the problem to surface?
- Do any other applications fail at the same time?

Answering these types of questions can help you explain the environment in which the problem occurs and correlate any dependencies. Remember that just because multiple problems might have occurred around the same time, the problems are not necessarily related.

Can the problem be reproduced?

From a troubleshooting standpoint, the “ideal” problem is one that can be reproduced. Typically with problems that can be reproduced, you have a larger set of tools or procedures at your disposal to help you investigate. Consequently, problems that you can reproduce are often simpler to debug and solve. However, problems that you can reproduce can have a disadvantage: If the problem is of significant business impact, you do not want it to recur! If possible, re-create the problem in a test or development environment, which typically offers you more flexibility and control during your investigation.

Tip: Simplify the scenario to isolate the problem to a suspected component.

The following questions can help you with reproducing the problem:

- Can the problem be re-created on a test machine?

- Are multiple users or applications encountering the same type of problem?
- Can the problem be re-created by running a single command, a set of commands, a particular application, or a stand-alone application?

Chapter 15. Troubleshooting checklist for WebSphere ESB

Asking questions about hardware and software requirements, product fixes, specific problems, error messages, and diagnostic data can help you troubleshoot WebSphere ESB.

The following questions can help you to identify the source of a problem that is occurring with WebSphere ESB:

1. Is the configuration supported?

   Refer to the requirements for WebSphere ESB to ensure that your system meets all hardware, operating system, and software requirements: http://www-1.ibm.com/support/docview.wss?uid=swg27006912.
2. Have you applied the latest fixes?
3. What is the problem?
   - Installing and configuring WebSphere ESB
   - Failed to log in to Process Center.
   - Deploying applications on WebSphere ESB
   - Administering applications and components on WebSphere ESB
   - Using WebSphere Application Server capabilities in WebSphere ESB
4. Have any error messages been issued? Look up the description and action for the messages in Reference > Messages.
5. For additional help in finding error and warning messages, interpreting messages, and configuring log files, see Diagnosing problems with message logs in the Websphere Application Server information center.
6. Difficult problems can require the use of tracing, which exposes the low-level flow of control and interactions between components. For help in understanding and using traces, see Working with trace in the Websphere Application Server information center.
7. If the checklist does not guide you to a resolution, you can collect additional diagnostic data. This data is necessary for IBM Support to effectively troubleshoot and assist you in resolving the problem. For more information, see Chapter 29, “Contacting IBM Software Support,” on page 1131.

© Copyright IBM Corp. 2007, 2011

Chapter 16. Messages overview

When you receive a message from WebSphere ESB, you can often resolve the problem by reading the entire message text and the recovery actions that are associated with the message.

You can find the full text of runtime messages, their explanations, and the recommended recovery actions by searching for the message identifier in the Messages section of the WebSphere ESB Reference documentation.

Runtime message identifiers consist of a four- or five-character message prefix, followed by a four- or five-character message number, followed by a single-letter message type code. For example, zzzzL1042C. The message type code describes the severity of the error message:

C Indicates a severe message.

E Indicates an urgent message.

I Indicates an informational message.

N Indicates an error message.

W Indicates a warning message.

Related information:

Messages: installation and profile creation
Some of the most commonly found error messages encountered when installing and configuring can be addressed with actions that resolve the underlying problems.

© Copyright IBM Corp. 2007, 2011

Chapter 17. WebSphere ESB log files

WebSphere ESB offers a comprehensive set of log files to help you identify and resolve problems during installation, configuration, and run time.

Installation and configuration logs

Various log files are created during the installation and uninstallation of WebSphere ESB and during profile creation, augmentation, and deletion. Examine these logs when problems occur during the product installation and configuration process.

Runtime logs

By default, log, error, and trace information for all processes and applications on a process server is written to the `SystemOut.log`, `SystemErr.log`, and `trace.log` files, which are stored in the *install_root*/profiles/*profile_name*/logs/*server_name* directory. These files are standard log files provided by WebSphere Application Server and capture information based on how logging and tracing are configured for the server.

Typically, the `SystemOut.log` file is used to monitor the health of the running server and for problem determination; entries include information to help you identify the specific component associated with the message. The `SystemErr.log` file contains exception stack trace information that is useful when performing problem analysis. When tracing is enabled, the `trace.log` file contains trace messages that can be used for problem determination.

Note: Content from the following log files is now captured in the `SystemOut.log` file:

- TW BPD Engine: Errors generated as a result of process instance execution on the current server
- TW Console: Actions that occurred in the Process Admin console
- TW Error: Java exceptions
- TW EventManager: Historical information about Event Manager processing
- TW Exp/Imp:Process export and import transactions in Process Designer
- TW Limit: Process server limit overruns
- TW JavaScript: Logging associated with JavaScript log functions like log.info() or log.debug()
- WS Inbound: Calls to published web services
- WS Outbound: Data about web services consumed by processes
- WS UCA Execution: Errors generated by Undercover Agent (UCA) execution

© Copyright IBM Corp. 2007, 2011

Related concepts:

"Profile-specific log files" on page 1059
There are log files detailing the characteristics and runtime activities of individual profiles. These log files are located within the profile directory for each profile.

Related information:

Tracing and logging configuration

Log and trace settings

Installation and profile creation log files
Various log files are created during installation and uninstallation of WebSphere ESB and during profile creation, augmentation, and deletion. Consult the applicable logs if problems occur during these procedures.

Chapter 18. Transaction log file

The transaction (tranlog) log file stores critical transactional data that is written to databases. It is an internal file that WebSphere Application Server uses to manage in-flight transactions and attempt to recover them if the server locks up.

CAUTION: Never delete the transaction log file from a production environment. Deleting this file removes information on in-flight transactions from WebSphere ESB memory. Without the transaction log file, there is no functionality to recover transactional information. In addition, long-running processes remain in an inconsistent state and you cannot complete the process flow except by deleting running instances. Deleting running instances might cause you to lose operational or business-critical data, which makes the database inconsistent with the message destination. Other inconsistencies that may be caused by deleting the transaction log file includes the following:

- Started transactions will neither be rolled back nor committed
- Artifacts will remain in the Java Virtual Machine (JVM) since they are referenced or allocated by a transaction but never garbage collected
- Service Component Architecture (SCA) messages that belong to a process navigation and transaction remain on SCA-related queues

Deleting the transaction log from a development environment causes the same problems. Because you can recreate business processes, deleting the files from a test environment is not as damaging as deleting them from a production environment.

© Copyright IBM Corp. 2007, 2011

Chapter 19. Troubleshooting installation and configuration

You can diagnose problems when the installation and configuration of WebSphere ESB is unsuccessful.

Procedure

1. Read any error messages from the installation process.

   See the following topic for an explanation: Error messages: installation and profile creation and augmentation. If the message corresponds to any of those described, correct the problem, clean up any installed portions, and try to reinstall.
2. If the installation of WebSphere Application Server Network Deployment was not successful, check Troubleshooting installation in the WebSphere Application Server Network Deployment information center and use the information found there to correct the problem before attempting to reinstall WebSphere ESB.
3. If the installation of WebSphere Application Server Feature Pack for Service Component Architecture (SCA) with the Service Data Objects (SDO) feature not successful, check Troubleshooting installation in the WebSphere Application Server Network Deployment information center and use the information found there to correct the problem before attempting to reinstall WebSphere ESB.
4. If the installation of WebSphere Feature Pack for Web Services was not successful (and installation of WebSphere Application Server Network Deployment was), check Troubleshooting Web server plug-ins installation and removal in the WebSphere Application Server Network Deployment information center and use the information found there to correct the problem before attempting to reinstall WebSphere ESB.

   Tip: If a problem occurs during an installation of WebSphere Feature Pack for Web Services as part of the WebSphere ESB installation, the installation process will not continue and an error message will be displayed.
5. If the installation of WebSphere ESB was not successful (and installation of WebSphere Application Server Network Deployment and WebSphere Feature Pack for Web Services were), check other WebSphere ESB installation log files. For information about the names, locations, and descriptions of these log files, see Installation and profile creation log files.
6. If you have successfully created a server profile, use the First steps console or the command-line method to start the server.
7. Verify that the server starts and loads properly by looking for a running Java process and the *Open for e-business* message in the `SystemOut.log` and `SystemErr.log` files.

   If no Java process exists or if the message is not displayed, examine the same logs for any miscellaneous errors. Correct any errors and try again.

   You can find the `SystemOut.log` and `SystemErr.log` files in the following platform-specific directories:

   - Linux UNIX **On Linux and UNIX platforms:** *profile_root*`/logs/`*servername*
   - Windows **On Windows platforms:** *profile_root*`\logs\`*servername*

© Copyright IBM Corp. 2007, 2011

8. Use the First steps console or the command-line method to stop the server, if it is running.
9. If you want to use a Snoop Servlet to verify the ability of the Web server to retrieve an application from WebSphere ESB, see the step "Start the Snoop servlet to verify the ability of the Web server to retrieve an application from the Application Server" in Troubleshooting installation in the WebSphere Application Server Network Deployment documentation.
10. Start the administrative console. For more information, see Starting and stopping the administrative console.
11. To resolve any IP address caching problems, see the step about resolving any IP address caching problems in Troubleshooting installation in the WebSphere Application Server Network Deployment documentation.

What to do next

On the product support Web site, you can review current information about resolutions to known problems, and you can read documents that can save you time gathering the information that you need to resolve a problem. Before opening a PMR, see the IBM WebSphere ESB support page.

Messages and known issues during installation and profile creation

Some of the most commonly found error messages encountered when installing and configuring can be addressed with actions that resolve the underlying problems.

Note: Linux UNIX Windows The following WebSphere ESB installation and configuration errors appear on Linux, UNIX, and Windows platforms.

If you do not see an error message that resembles yours, or if the information provided does not solve your problem, contact WebSphere ESB support at IBM for further assistance.

Informational messages

The following messages are for information only and do not require any action:

- `Sample applications feature is not available for installation because the WAS Samples feature is not installed.`
- `Installation Manager cannot remove feature import.productProviders.feature from an installation package that was imported to Installation Manager.`
- `Installation Manager cannot remove feature import.configLauncher.feature from an installation package that was imported to Installation Manager.`

Configuration errors were detected during the installation

If you are installing either the Advanced or Advanced Process Server editions of IBM Business Process Manager V7.5.1 with Installation Manager, and you selected to install only the client, you might get the following warning message: `The packages are installed with warnings. View Log file`

If you see this warning message after installing the client feature of Business Process Manager, either the Advanced or the Advanced Process Server edition, on a Windows system with just the client feature selected, examine the logs. If, in the logs, you see the following message, then you cannot use the Windows add or

remove programs function to remove the client: `2464 WARNING 27:29.66 com.ibm.ws.exec.command.ExecCommand "C:\IBM\WebSphere\AppServer/util/bpm_configManagerLauncher.bat" cannot be found`

To remove the client, you must use Installation Manager.

Supported IBM JDK was not found. The IBM JDK shipped with this product must be located at *install_root*/JDK. Correct this problem and try again.

If you use symbolic links to point to the IBM Java Development Kit (JDK) shipped with the product, or to a JDK found in the PATH environment variable on your system, IBM SDK for Java validation might fail, resulting in a failed installation. This problem is caused by the way IBM SDK for Java validation code detects whether the JDK shipped with the product is the current JDK used for installation.

To resolve this problem, do not use symbolic links in JVMs supplied with the installation image of WebSphere ESB and remove symbolic links from all JVMs that appear in your system's PATH environment variable.

Warning: Cannot convert string “<type_name>” to type FontStruct

If you install the Web server plug-ins for WebSphere Application Server, you also install the ikeyman utility. The ikeyman utility is part of the Global Services Kit 7 (GSKit7).

Linux If you issue the `ikeyman.sh` script on a Linux system, you might see the following message:

```
Warning: Cannot convert string
   "-monotype-arial-regular-r-normal--*-140-*-*-p-*-iso8859-1"
   to type FontStruct
```

You can safely ignore the warning and use the ikeyman utility.

CWWBB0627E error during installation with DB2 Express on Linux systems

Linux

When you install IBM Business Process Manager, IBM Business Monitor, or WebSphere Enterprise Service Bus with DB2 Express® on a 32- or 64-bit Linux system, the server does not start successfully. Also, the `SystemOut.log` file includes `CWWBB0627E: Failed to create the database schema`.

The problem occurs if you used the Administration server (DAS) user name, which defaults to `bpmadmin`, instead of the Instance user name, which defaults to `bpminst`.

This problem can occur in the following situations:

- When you use the Administration server (DAS) user name during a Custom installation, in which you create the profile after installation
- During a Typical installation with an existing DB2 database

If the problem occurs, the `SystemOut.log` file contains the following information:

```
[5/24/11 10:40:27:131 CDT] 00000000 CreateSchemaM E   CWWBB0627E: Failed to create the database schema.
[5/24/11 10:40:27:227 CDT] 00000000 ProcessEngine E
[5/24/11 10:40:27:274 CDT] 00000000 ManagerAdmin  I   TRAS0018I: The trace state has changed. The new trace state is *=info.
[5/24/11 10:40:27:499 CDT] 00000000 CreateSchemaM I   CWWBB0625I: Started creating the database schema.
[5/24/11 10:40:27:502 CDT] 00000000 CreateSchemaM I   CWWBB0658I: Schema qualifier is: 'BPEDB'.
[5/24/11 10:40:27:909 CDT] 00000000 CreateSchemaM I   CWWBB0614E: Database schema creation or migration step failure
  CREATE TABLE BPEDB.PROCESS_TEMPLATE_B_T ( PTID CHAR(16) FOR BIT DATA NOT NULL , NAME VARCHAR(220) NOT NULL ,
  DEFINITION_NAME VARCHAR(220) , DISPLAY_NAME VARCHAR(64) , APPLICATION_NAME VARCHAR(220) , DISPLAY_ID INTEGER
  NOT NULL , DISPLAY_ID_EXT VARCHAR(32) , DESCRIPTION VARCHAR(254) , DOCUMENTATION CLOB(4096) , EXECUTION_MODE
  INTEGER NOT NULL , IS_SHARED SMALLINT NOT NULL , IS_AD_HOC SMALLINT NOT NULL , STATE INTEGER NOT NULL ,
  VALID_FROM TIMESTAMP NOT NULL , TARGET_NAMESPACE VARCHAR(250) , CREATED TIMESTAMP NOT NULL , AUTO_DELETE
  SMALLINT NOT NULL , EXTENDED_AUTO_DELETE INTEGER NOT NULL , VERSION VARCHAR(32) , SCHEMA_VERSION INTEGER NOT
  NULL , ABSTRACT_BASE_NAME VARCHAR(254) , S_BEAN_LOOKUP_NAME VARCHAR(254) , S_BEAN60_LOOKUP_NAME VARCHAR(254) ,
  E_BEAN_LOOKUP_NAME VARCHAR(254) , PROCESS_BASE_NAME VARCHAR(254) , S_BEAN_HOME_NAME VARCHAR(254) ,
  E_BEAN_HOME_NAME VARCHAR(254) , BPEWS_UTID CHAR(16) FOR BIT DATA , WPC_UTID CHAR(16) FOR BIT DATA , BPMN_UTID
  CHAR(16) FOR BIT DATA , BUSINESS_RELEVANCE SMALLINT NOT NULL , ADMINISTRATOR_QTID CHAR(16) FOR BIT DATA ,
  READER_QTID CHAR(16) FOR BIT DATA , A_TKTID CHAR(16) FOR BIT DATA , A_TKTIDFOR_ACTS CHAR(16) FOR BIT DATA ,
  COMPENSATION_SPHERE INTEGER NOT NULL , AUTONOMY INTEGER NOT NULL , CAN_CALL SMALLINT NOT NULL , CAN_INITIATE
  SMALLINT NOT NULL , CONTINUE_ON_ERROR SMALLINT NOT NULL , IGNORE_MISSING_DATA INTEGER NOT NULL , EAR_VERSION
  INTEGER NOT NULL , LANGUAGE_TYPE INTEGER NOT NULL , DEPLOY_TYPE INTEGER NOT NULL , MESSAGE_DIGEST VARCHAR(20)
  FOR BIT DATA , CUSTOM_TEXT1 VARCHAR(64) , CUSTOM_TEXT2 VARCHAR(64) , CUSTOM_TEXT3 VARCHAR(64) , CUSTOM_TEXT4
  VARCHAR(64) , CUSTOM_TEXT5 VARCHAR(64) , CUSTOM_TEXT6 VARCHAR(64) , CUSTOM_TEXT7 VARCHAR(64) , CUSTOM_TEXT8
  VARCHAR(64) , PRIMARY KEY ( PTID ) ) IN BPETS8K: com.ibm.db2.jcc.am.SqlSyntaxErrorException: DB2 SQL Error:
  SQLCODE=-204, SQLSTATE=42704, SQLERRMC=BPETS8K, DRIVER=3.61.65.
[5/24/11 10:40:27:912 CDT] 00000000 CreateSchemaM E   CWWBB0627E: Failed to create the database schema.
[5/24/11 10:40:27:912 CDT] 00000000 CreateSchemaM E   CWWBB0627E: Failed to create the database schema.
[5/24/11 10:40:27:948 CDT] 00000000 TraceBPE      E
```

To solve the problem, you can either create a new profile, or you can use the `usermod` command to add the Administrative server (DAS) user name:

- Create a new profile using the **-manageprofiles** command. Specify the Instance user name, which, by default, is `bpminst`, for the **-dbUserId** parameter.
- Add the Administrative server (DAS) user name, which, by default, is `bpmadmin`, to the `bpmiadm1` group by running the following command:

  ```
  usermod -a -G bpmiadm1 bpmadmin
  ```

Installation and profile creation log files

Various log files are created during installation and uninstallation of WebSphere ESB and during profile creation, augmentation, and deletion. Consult the applicable logs if problems occur during these procedures.

Table 110 on page 1043 shows the log file names, locations, and descriptions for success and failure for WebSphere ESB.

Some directory paths, file names, and indicator values in Table 110 on page 1043 contain spaces to allow the entries to fit in the table cells. The actual directory paths, file names, and indicator values do not contain spaces.

The variable *install_root* represents the installation directory of WebSphere ESB. The variable *profile_root* represents the root location of a profile.

For more information see Installation directories for the product and profiles.

Table 110. Installation and profile logs for WebSphere ESB components

Log name and location	Log description
Agent data location/logs Typically: C:\Documents and Settings\All Users\Application Data\IBM\Installation Manager\logs*Agent data location*/logs Typically: /var/ibm/InstallationManager/logs	Installation Manager log file directory under the *Agent data location*. For more information on the *Agent data location* refer to the Installation Manager documentation. Contains log information for Business Process Manager and WebSphere Application Server installations and uninstallations.
• Linux UNIX *install_root*/logs/wbi/install/installconfig_server.log • Windows *install_root*\logs\wbi\install\installconfig_server.log	Logs configuration actions that run at the end of the installation process to configure components, install system applications, and create Windows shortcuts and registry entries.
• Linux UNIX *install_root*/logs/manageprofiles/pmt.log • Windows *install_root*\logs\manageprofiles\pmt.log	Logs all events from the Profile Management Tool.
• Linux UNIX *install_root*/logs/manageprofiles/*profile_name*_create.log • Windows *install_root*\logs\manageprofiles*profile_name*_create.log	• Traces all events that occur during the creation of the named profile. • Created when a profile is created during a typical installation, custom installation (only Advanced PS), when using the Profile Management Tool, or when using the manageprofiles command-line utility.
• Linux UNIX *install_root*/logs/manageprofiles/*profile_name*_create_error.log • Windows *install_root*\logs\wbi\update*profile_name*_create_error.log	Logs information extracted from the*profile_name*_create.log file. This information pertains to any failing configuration actions, validations, wsadmin calls and or any corresponding log files.
• Linux UNIX *install_root*/logs/manageprofiles/*profile_name*_augment.log • Windows *install_root*\logs\manageprofiles*profile_name*_augment.log	• Traces all events that occur during the augmentation of the named profile. • Created when a profile is augmented, when using the Profile Management Tool, or when using the **manageprofiles** command-line utility.
• Linux UNIX *install_root*/logs/manageprofiles/*profile_name*_augment_error.log • Windows *install_root*\logs\wbi\update*profile_name*_augment_error.log	Logs information extracted from the *profile_name*_augment.log file. This information pertains to any failing configuration actions, validations, wsadmin calls and any corresponding log files.
• Linux UNIX *install_root*/logs/manageprofiles/*profile_name*_delete.log • Windows *install_root*/logs/manageprofiles/*profile_name*_delete.log	• Traces all events that occur during the deletion of the named profile. • Created when profile deletion is performed with the **manageprofiles** command-line utility.
• Linux UNIX *install_root*/logs/wbi/uninstall/uninstallconfig_server.log • Windows *install_root*\logs\wbi\uninstall\uninstallconfig_server.log	Logs all configuration actions that run during uninstallation events relating to IBM Business Process Manager.

Launching Installation Manager directly on 64-bit systems

You can launch Installation Manager directly, for WebSphere ESB, IBM Business Monitor, or IBM WebSphere Enterprise Service Bus, on 64-bit systems.

About this task

To launch Installation Manager directly, go to *image_location*/IM and do one of the following:

- If you do not have Installation Manager installed, run: `install -input install_64.xml`
- If you do have Installation Manager installed, run: `install -input post-install_64.xml`

This will start Installation Manager with the correct products preselected.

Warnings about GTK or ulimit on Linux or UNIX when installing or migrating

On the Linux or UNIX operating system, when you are installing or migrating, you might see a warning about 32-bit GTK libraries or increasing your ulimit.

GTK warning

If you are on a 64-bit system, you might receive the following message:

```
Your operating system failed the launchpad prerequisites check. The following 32-bit GTK Library for running IBM Installation Manager is not available in underlying OS: list_of_missing_files. Please install the 32-bit GTK Library and restart your installation.
```

If you see this message, your server does not have the 32-bit version of the GTK library installed, or the library is an incorrect version. You must update your server with the correct version of the 32-bit GTK library before you continue the installation. You can get the library from the DVD or official web site of your operating system.

ulimit warning

If you receive the following warning message during the prerequisite checking, use the platform-specific steps below to increase the `ulimit` number.

```
Current system has detected a lower level of ulimit than the recommended value of recommended_value. Please increase the ulimit number to minimum value of recommended_value and re-start the installation.

Shutdown your installer. If you are a root user open a command prompt and issue ulimit -n recommended_value and then restart the installer. If you are a non-root user, work with your system administrator to increase your ulimit -n recommended_value and then restart the installer.
```

The required value is calculated based on the version of WebSphere Application Server, the feature packs, and the configuration that you are installing.

1. Set the maximum number of open files using the following steps: Linux
   a. Open `/etc/security/limits.conf`.
   b. Locate the `nofile` parameter and increase the value. If a line containing the `nofile` parameter does not exist, add the following lines to the file:
      `* hard nofile` *recommended_value*
      `* soft nofile` *recommended_value*
   c. Save and close the file.
   d. Log off and log in again.
2. Restart the computer.
3. Restart the installer.

Troubleshooting the launchpad application or First Steps

If the launchpad application or First Steps does not start, try the following troubleshooting tips.

Troubleshooting the launchpad application

Restart the launchpad after you make any changes.

- If you are using images from Passport Advantage®, make sure that you extract the contents of the images for *WebSphere Enterprise Service Bus Version 7.5* DVD, *WebSphere Application Server Network Deployment Supplement Version 7.5* CDs, and *WebSphere Portal add-in for WebSphere Enterprise Service Bus* DVD into separate directories. Extracting the files from the images into the same directory will cause errors to occur. It is recommended that you use sibling directories. For example, use a set of directories such as the following:
- If you can start the launchpad, but selecting a link does not resolve to a page in the launchpad, you might have the media for the wrong operating system in the disk drive. Check the validity of the media.
- Windows If you are attempting to use the Mozilla browser on a Windows system, Internet Explorer might open instead. The launchpad does not recognize Mozilla as the default browser if Internet Explorer is also installed on the same system. The launchpad is fully functional with Internet Explorer, so no action is required.

  To create an environment variable that forces the use of Mozilla, issue the following case-specific command at a command prompt:

  ```
  set BROWSER=Mozilla
  ```
- Ensure that the JavaScript function is enabled in your browser.

  Linux UNIX Mozilla: Click **Edit > Preferences > Advanced > Scripts & Plugins**:
  - Enable JavaScript for: Navigator
  - Allow scripts to ... (Select all boxes)

  Linux UNIX Mozilla Firefox: Click **Tools > Options > Content**:
  - Select **Enable Java**
  - Select **Enable JavaScript**
  - Click **Advanced** and Allow scripts to ... (Select all boxes)

  Windows Internet Explorer: Click **Tools > Internet Options > Security > Custom Level for Internet > Scripting > Active scripting > Enable**

Troubleshooting First Steps

If links from the First Steps console fail to open browser windows, or the First Steps console fails to launch, or immediately quits, on the Microsoft Windows operating system when Mozilla Firefox is set as the default browser, try the following workarounds.

- Modify the Windows registry to delete the spaces in the location name:
  1. Navigate to `HKEY_LOCAL_MACHINE\SOFTWARE\Clients\StartMenuInternet\FIREFOX.EXE\shell\open\command`.

     Note: The preceding line might be word-wrapped. Be sure to navigate to the location specified in the preceding lines, up to the "`command`" key in the registry.

2. Change the "(Default)" entry so that spaces are removed from the path.

   For example, if the path is set as "`C:\Program Files\Mozilla Firefox\firefox.exe`", change the path to its short equivalent "`C:\Progra~1\Mozill~1\firefox.exe`".

   The short names might not be the same on all systems. For example, if you have installed "Mozilla Thunderbird" as well as "Mozilla Firefox", and both are installed in the "Program Files" directory, the short name to the location of Mozilla Firefox might be different from that in the example above. You can use the "`dir /X`" command to determine the short names of individual files and directories located in the current directory.

   Note: If you choose this option, be careful that you do not corrupt the Windows registry. This key might vary for different locales, so use caution or choose another workaround. It is recommended that you back up the registry before making any changes.

- Install Mozilla Firefox to a different location that does not contain spaces.
- Change the default browser temporarily.
  1. Set Windows Internet Explorer as the default browser.
  2. Reset Mozilla Firefox as the default browser. This automatically changes the registry entry in the first workaround so that the spaces are removed.

     This only works when you set the default browser from within the Mozilla Firefox application. It will not work when using the "*Set Program Access and Defaults*" command in "*Add/Remove Programs*".

First Steps console fails to start on the Windows operating system

To run First Steps on Windows 7, Windows Vista, or Windows Server 2008, you must elevate your Microsoft Windows user account privileges by right-clicking the `firststeps.bat` file and selecting **Run as administrator**. This is required for both administrative and nonadministrative users.

By default, the `firststeps.bat` file is located in:

profile_root`/firststeps/bpmPS/`

Diagnosing a failing Ant configuration script

Determine whether a product installation problem on an operating system is caused by a failing Apache Ant configuration script.

Before you begin

Start diagnosing installation problems by looking at the troubleshooting procedure. See Troubleshooting installation and configuration. After the installation completes successfully, several Ant scripts configure the product. The following procedure describes what to do when an Ant script fails. When the installation log does not indicate a failure, determine how to correct any problems with failing Ant configuration scripts.

About this task

The *install_root*`/logs/wbi/install/installconfig_server.log` file, when present, describes any failure of an Ant script. Determine if any of the following configuration scripts failed. If so, use the configuration script recovery procedures.

Use the investigative action to manually verify that the following configuration scripts ran successfully during the configuration of the WebSphere ESB product. If any script failed, use the recovery action steps to complete the function of the script.

To diagnose failed Ant configuration scripts, perform the following steps.

Procedure

- Diagnose the failed `90SConfigWBIMigrationScript.ant` configuration script. This script changes the permissions of the following script to 755: *install_root*/bin/BPMMigrate. This script also replaces the following tokens in the *install_root*/bin/BPMMigrate script:

From:	To the value that you selected during installation:
${JAVAROOT}	*install_root*/java/jre/bin/java
${MIGRATIONJAR}	*install_root*/bin/migration/migrationGUI/migrationGUI.jar
${WASROOT}	*install_root*
${PRODUCTID}	${WS_CMT_PRODUCT_TYPE}

  1. Investigative action: Verify that the permissions are 755 for the following directories:
     - Linux UNIX *install_root*/bin/BPMMigrate.sh
     - Windows *install_root*\bin\BPMMigrate.bat
  2. Recovery action: Issue the following command:
     - Linux UNIX chmod 755 *install_root*/bin/BPMMigrate.sh
     - Windows chmod 755 *install_root*\bin\BPMMigrate.bat
  3. Investigative action: Open the following file in an editor and verify that real values exist instead of the following values: ${JAVAROOT}, ${MIGRATIONJAR}, ${WASROOT}, and ${PRODUCTID}.
     - Linux UNIX *install_root*/bin/BPMMigrate.sh
     - Windows *install_root*\bin\BPMMigrate.bat
  4. Recovery action: Change the following tokens to values in the BPMMigrate script: ${JAVAROOT}, ${MIGRATIONJAR}, ${WASROOT}, and ${PRODUCTID}.
- Diagnose the failed `85SConfigNoProfileFirstStepsWBI.ant`. This script copies all files from the *install_root*/properties/version/install.wbi/firststeps.wbi directory to the *install_root*/firststeps/wbi/html/noprofile directory. This script also replaces the following tokens in the following files:
  - Linux UNIX *install_root*/firststeps/wbi/firststeps.sh
  - Windows *install_root*\firststeps\wbi\firststeps.bat

From:	To the value that you selected during installation:
${JAVAROOT}	*install_root*/java/jre/bin/java
${PROFILEROOT}	*install_root*
${HTMLSHELLJAR}	*install_root*/lib/htmlshellwbi.jar
${CELLNAME}	${WS_CMT_CELL_NAME}

1. Investigative action: Verify that all files are copied from the *install_root*/properties/version/install.wbi/firststeps.wbi directory to the *install_root*/firststeps/wbi/html/noprofile directory.
2. Recovery action: Copy all of the files from the *install_root*/properties/version/install.wbi/firststeps.wbi directory to the *install_root*/firststeps/wbi/html/noprofile directory.
3. Investigative action: Open the *install_root*/firststeps/wbi/firststeps script in an editor. Verify that real values exist instead of the following values: ${JAVAROOT}, ${PROFILEROOT}, ${HTMLSHELLJAR}, and ${CELLNAME}.
4. Recovery action: Change the following tokens to values in the *install_root*/firststeps/wbi/firststeps script. ${JAVAROOT}, ${PROFILEROOT}, ${HTMLSHELLJAR}, and ${CELLNAME}.

Results

After you correct any installation errors and any Ant script configuration errors by performing the corrective actions in this procedure, the installation is complete.

What to do next

Start the First steps console.

Profile creation fails after reinstallation with DB2 Express

When you reinstall IBM Business Process Manager, IBM Integration Designer, or WebSphere Enterprise Service Bus with DB2 Express, the profile creation step fails when it tries to create the databases.

When you uninstall the product, the uninstallation program removes DB2 Express, but it does not remove the BPMINST directory, which contains the databases. During reinstallation, the profile creation step fails because the databases already exist.

To resolve the problem, delete the BPMINST directory before reinstalling the server images.

Note: The location of the BPMINST directory varies, depending on the operating system that you are using.

- On a Windows operating system, the location of the directory depends on which drive you used for installation. The directory is located at: <*drive*>:\BPMINST (for example, C:\BPMINST or D:\BPMINST).
- On a Linux operating system, the location of the directory depends on whether you changed the instance user ID.
  - If you did not change the ID, the directory is located at: /home/bpminst/bpminst.
  - If you changed the ID, the directory is located at: /home/<*bpm_instance_userid*>/bpminst.

Profile creation fails on Windows operating system

The Profile Management Tool or **manageprofiles** command on WebSphere ESB might fail on the Microsoft Windows operating system if it is launched without administrative privilege.

To install or run the Profile Management Tool on Windows 7, Windows Vista, or Windows Server 2008, you must elevate your Microsoft Windows user account privileges. If you do not use the correct privileges the profile creation will fail and you will receive an SQL1092N message in the log which indicates that the user does not have authority to perform the requested actions. Whether you are an administrative user or a non-administrative user, right-click the `pmt.bat` file and select **Run as administrator**.

Alternatively, use the **runas** command at the command line. For example, the following command can run be executed from the *install_root*\bin\ ProfileManagement directory:

```
runas /user:MyAdminName /env pmt.bat
```

Non-administrative users are prompted for the administrator password.

To install or run the **manageprofiles** command on Windows 7, Windows Vista, or Windows Server 2008, you must elevate your Microsoft Windows user account privileges using the **runas** command. Remember to put double quotation marks around the **manageprofiles** command and all parameters. For example, the following command can run be executed from the *install_root*\bin directory:

Non-administrative users are prompted for the administrator password.

Successful installation reported after profile creation failure

If the profile creation step fails during a custom installation, the failure is not recognized by Installation Manager, which reports a successful installation. This problem occurs only on Windows platforms.

This problem applies to IBM Business Process Manager - Process Server, IBM Business Monitor, or WebSphere Enterprise Service Bus.

The failure of the profile creation step is indicated in the profile creation log. For example, you might see the following error: `INSTCONFFAILED: Profile augmentation failed.`

To resolve the problem, perform the following steps:

1. Check the profile creation log, which is located in the `<install_root>/logs/manageprofiles` directory, to determine the cause of the error.
2. Complete one of the following steps:
   - Delete the profile that contains the error. Then, use the **manageprofiles** command or the Profile Management Tool to create a new profile.
   - Install the product with a custom installation, but do not create a profile. Then, after installation, use the **manageprofiles** command or the Profile Management Tool to create the profile.

Recovering from profile creation or augmentation failure

The Profile Management Tool can experience failures when creating new or augmenting existing profiles. The same can occur using the **manageprofiles** command-line utility. If such a failure occurs, first check the log files as described in this topic, then follow the recovery instructions described, depending on the situation.

Log files

All manageprofiles log files are in *install_root*/logs/manageprofiles. Look at the following log files in the order given. Each log file must contain the entry "INSTCONFSUCCESS." If a file does not include this entry, a failure was detected. Look at the log files to determine why a failure was encountered and to determine a remedy.

1. The log file *profile_name*_create_error.log (where *profile_name* is the name of the profile).

   Note: Look at this file only if you were creating a new profile, not augmenting an existing one.

   - Linux UNIX *install_root*/logs/manageprofiles/*profile_name*_create_error.log
   - Windows *install_root*\logs\wbi\update*profile_name*_create_error.log

   Search for the text Configuration action succeeded or Configuration action failed.

   Note: There can be multiple occurrences of Configuration action failed. Investigate and remedy each one. Also review the log files described in the following options, if the profile was created.

   Note: Additional information is available in the manageprofiles directory in the pmt.log, which logs all events that occur when a default profile is created during a complete installation using the Profile Management Tool.
2. Log file *profile_name*_augment_error.log (where *profile_name* is the name of the profile).

   This log file is located in the following directories:

   - Linux UNIX *install_root*/logs/manageprofiles/*profile_name*_augment_error.log
   - Windows *install_root*\logs\wbi\update*profile_name*_augment_error.log

   Search for the text Configuration action succeeded or Configuration action failed.

   Note: There can be multiple occurrences of Configuration action failed. Investigate and remedy each one. Also review the log files described in the following options, if the profile was created.

   Note: If you want to know the status of a profile you created during installation, run the following commands:

   - Linux UNIX *install_root*/bin/logProfileErrors.sh
   - Windows *install_root*\bin\logProfileErrors.bat
3. Individual profile template action log files.

   If you discovered false values in the log files described in the preceding options, review the log files in the following directories:

   - Linux UNIX *install_root*/logs/manageprofiles/*profile_name* on Linux and UNIX systems
   - Windows *install_root*\logs\manageprofiles*profile_name* on Windows systems

   where *profile_root* or *user_data_root* is the installation location of the profile.

These log files do not follow a consistent naming convention, but typically, each is the name of the Apache Ant script that failed followed by `.log`. For example, suppose the following entry is in the *profile_name*_augment.log file:

```
<messages>Result of executing
E:\o0536.15\profileTemplates\default.wbicore\actions\saveParamsWbiCore.ant
was:false</messages>
```

First look at the surrounding entries in the *profile_name*_augment.log file in the *install_root*/logs/manageprofiles directory. If you cannot determine the cause of the failure from the surrounding entries, look for the corresponding log file for any failing Ant script entries. In this case, the log file created by the `saveParamsWbiCore.ant` script is `saveParamsWbiCore.ant.log`. Look at that file to investigate why the failure occurred.

Recovery for creation failure

After you determine why profile creation failed and address the cause of the failure, you can try to create the profile again.

Note: When you create a profile, it first creates a WebSphere Application Server profile and then augments it with WebSphere ESB profile templates to create the WebSphere ESB profile. Even if you encountered a profile creation failure, a profile can exist that does not have all the needed augmentations.

To determine if the profile exists, run the *install_root*/bin/manageprofiles -listProfiles command. If the profile name you used for creation does not exist, you can re-create the profile. If the profile name you used for creation exists, then the profile was created and you have encountered an augmentation failure. For tips on recovering from an augmentation failure, see “Recovery for augmentation failure.”

Recovery for augmentation failure

After you determine why profile augmentation failed and address the cause of the failure, you can try to augment the existing profile again to successfully create a complete WebSphere ESB profile by following these steps:

1. Start the Profile Management Tool and, instead of creating a new profile, choose to augment an existing profile.
2. Choose the profile you were working with, and enter the correct information for it.

Note: Some of the augmentations might have completed successfully the first time you ran the Profile Management Tool. As a result, you might not see all of the panels that you saw the first time you tried to create the profile. This is because the Profile Management Tool detects which remaining augmentations must be completed and displays only the necessary panels.

Reinstallation cannot create new profile

If you try to reinstall the product to the same location, or if you try to reinstall after a failed uninstall, the installation might fail because a new profile cannot be created.

If databases were created for the test environment, the databases must be dropped before you can create a new profile.

If the databases are not automatically dropped during uninstall, you must drop them manually.

- For the qbpmaps profile, the default databases are QBPMDB, QPDWDB, and QCMNDB
- For the qesb profile, the default databases are ECMNDB and QECMNDB (one or both)
- For the qmwas profile, the default databases are MONITOR and COGNOSCS
- For the qmbpmaps profile, the default databases are QBPMDB, QPDWDB, QCMNDB, MONITOR, and COGNOSCS
- For the qmesb profile, the default databases are ECMNDB, QECMNDB, MONITOR, and COGNOSCS

Chapter 20. Failure trying to serialize an object that is not serializable in a migrated BPEL file

If a serialization failure occurs in a BPEL file generated by the migration, you might be able to modify it to prevent the failure from occurring.

Problem: A serialization failure occurs in a custom snippet node of a business process execution language (BPEL) file generated by migration because an attempt is made to serialize an object that is not serializable.

Cause: In WebSphere InterChange Server or WebSphere Business Integration Server Express, a Collaboration Template is compiled into a single Java class. In WebSphere ESB, each node in a BPEL file might compile into a separate Java class. In WebSphere InterChange Server or WebSphere Business Integration Server Express, a variable can be declared once and shared throughout the various steps of a Collaboration Template. To simulate that behavior in the migrated BPEL file, each variable used in a code snippet must be retrieved at the start of the snippet and saved at the end of the snippet. Variables defined in WebSphere InterChange Server or WebSphere Business Integration Server Express Port definitions become BPEL variables. These are retrieved into BusObj variables at the beginning of each snippet (if referenced in the snippet) and saved back to the BPEL variables at the end of each snippet. For example, a retrieval at the beginning of snippets looks like this:

```
BusObj tempBusObj = null;if (tempBusObj_var != null) { tempBusObj =
      new BusObj(tempBusObj_var); };
```

and a save at the end of snippets looks like this:

```
if (tempBusObj == null) { tempBusObj_var = null; } else { tempBusObj_var =
      tempBusObj.getBusinessGraph(); }
```

Other variables used in the WebSphere InterChange Server or WebSphere Business Integration Server Express snippet code are serialized and stored as a String in a BPEL variable named *CollabTemplateName_var*. These variables are deserialized at the beginning of each BPEL snippet, and then serialized and saved at the end of each BPEL Snippet that they are referenced in. For example, objects are retrieved like this:

```
BusObj tempBusObj = (BusObj)BaseCollaboration.deserialize
  (FrontEndCollab_var.getString("tempBusObj"));
```

and objects are saved like this:

```
FrontEndCollab_var.setString("tempBusObj", BaseCollaboration.serialize(tempBusObj));
```

If the type of the object being serialized is not serializable, then using serialize and deserialize will fail when the BPEL is run.

Solution: After migration, modify the BPEL file as follows:

- For any variable that is not Java-serializable, update the BPEL snippets to remove the serialization and deserialization statements. If the variable needs to be shared across snippets (instead of being re-created in each snippet) another method must be used to preserve the value of the variable across snippets.

© Copyright IBM Corp. 2007, 2011

- Manually define BPEL variables for variables of type BusObj that are not declared in the WebSphere InterChange Server or WebSphere Business Integration Server Express Port definitions but are used on Partner Invokes. This is a manual step because variables used during invokes in WebSphere ESB must be strongly typed, and the migration tools cannot accurately determine that type from the WebSphere InterChange Server or WebSphere Business Integration Server Express snippets.

  Note: The naming convention used by the migration tools is to add `_var` to the name of the variable in the snippet code when naming the BPEL variables. For example, for a variable called tempBusObj in the snippet code, the migration tools will create a BPEL variable named `tempBusObj_var`.
- For variables that must be declared manually as BPEL variables, change the BPEL snippet code so that it uses the "deserialize/serialize" method of preserving these variables rather than the "retrieve from/store into BPEL variable" method of preserving these variables.

Chapter 21. Troubleshooting a failed deployment

This topic describes the steps to take to determine the cause of a problem when deploying an application. It also presents some possible solutions.

Before you begin

This topic assumes the following things:
- You have a basic understanding of debugging a module.
- Logging and tracing is active while the module is being deployed.

About this task

The task of troubleshooting a deployment begins after you receive notification of an error. There are various symptoms of a failed deployment that you have to inspect before taking action.

Procedure

1. Determine if the application installation failed.

   Examine the `SystemOut.log` file for messages that specify the cause of failure. Some of the reasons an application might not install include the following:
   - You are attempting to install an application on multiple servers in the same Network Deployment cell.
   - An application has the same name as an existing module on the Network Deployment cell to which you are installing the application.
   - You are attempting to deploy Java EE modules within an EAR file to different target servers.

   Important: If the installation has failed and the application contains services, you must remove any SIBus destinations or JCA activation specifications created before the failure before attempting to reinstall the application. The simplest way to remove these artifacts is to click **Save > Discard all** after the failure. If you inadvertently save the changes, you must manually remove the SIBus destinations and JCA activation specifications (see Deleting SIBus destinations and Deleting JCA activation specifications in the Administering section).
2. If the application is installed correctly, examine it to determine if it started successfully.

   If the application did not start successfully, the failure occurred when the server attempted to initiate the resources for the application.
   a. Examine the `SystemOut.log` file for messages that will direct you on how to proceed.
   b. Determine if resources required by the application are available and/or have started successfully.

      Resources that are not started prevent an application from running. This protects against lost information. The reasons for a resource not starting include:
      - Bindings are specified incorrectly
      - Resources are not configured correctly
      - Resources are not included in the resource archive (RAR) file

© Copyright IBM Corp. 2007, 2011

- Web resources not included in the Web services archive (WAR) file

c. Determine if any components are missing.

The reason for missing a component is an incorrectly built enterprise archive (EAR) file. Make sure that the all of the components required by the module are in the correct folders on the test system on which you built the Java archive (JAR) file. “Preparing to deploy to a server” contains additional information.

3. Examine the application to see if there is information flowing through it.

Even a running application can fail to process information. Reasons for this are similar to those mentioned in step 2b on page 1055.

a. Determine if the application uses any services contained in another application. Make sure that the other application is installed and has started successfully.

b. Determine if the import and export bindings for devices contained in other applications used by the failing application are configured correctly. Use the administrative console to examine and correct the bindings.

4. Correct the problem and restart the application.

Deleting JCA activation specifications

The system builds JCA application specifications when installing an application that contains services. There are occasions when you must delete these specifications before reinstalling the application.

Before you begin

If you are deleting the specification because of a failed application installation, make sure the module in the Java Naming and Directory Interface (JNDI) name matches the name of the module that failed to install. The second part of the JNDI name is the name of the module that implemented the destination. For example in `sca/SimpleBOCrsmA/ActivationSpec`, **`SimpleBOCrsmA`** is the module name.

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as administrator or configurator to perform this task.

About this task

Delete JCA activation specifications when you inadvertently saved a configuration after installing an application that contains services and do not require the specifications.

Procedure

1. Locate the activation specification to delete.

The specifications are contained in the resource adapter panel. Navigate to this panel by clicking **Resources > Resource adapters**.

a. Locate the **Platform Messaging Component SPI Resource Adapter.**

To locate this adapter, you must be at the **node** scope for a standalone server or at the **server** scope in a deployment environment.

2. Display the JCA activation specifications associated with the Platform Messaging Component SPI Resource Adapter.

Click on the resource adapter name and the next panel displays the associated specifications.

3. Delete all of the specifications with a **JNDI Name** that matches the module name that you are deleting.
   a. Click the check box next to the appropriate specifications.
   b. Click **Delete.**

Results

The system removes selected specifications from the display.

What to do next

Save the changes.

Deleting SIBus destinations

Service integration bus (SIBus) destinations are used to hold messages being processed by SCA modules. If a problem occurs, you might have to remove bus destinations to resolve the problem.

Before you begin

If you are deleting the destination because of a failed application installation, make sure the module in the destination name matches the name of the module that failed to install. The second part of the destination is the name of the module that implemented the destination. For example in `sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer`, **`SimpleBOCrsmA`** is the module name.

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as administrator or configurator to perform this task.

About this task

Delete SIBus destinations when you inadvertently saved a configuration after installing an application that contains services or you no longer need the destinations.

Note: This task deletes the destination from the SCA system bus only. You must remove the entries from the application bus also before reinstalling an application that contains services (see Deleting JCA activation specifications in the Administering section of this information center.

Procedure

1. Log into the administrative console.
2. Display the destinations on the SCA system bus.
   a. In the navigation pane, click **Service integration** > **buses**
   b. In the content pane, click **SCA.SYSTEM.*cell_name*.Bus**
   c. Under Destination resources, click **Destinations**
3. Select the check box next to each destination with a module name that matches the module that you are removing.
4. Click **Delete.**

Results

The panel displays only the remaining destinations.

What to do next

Delete the JCA activation specifications related to the module that created these destinations.

Chapter 22. Troubleshooting administration tasks and tools

Use the information in this group of topics to identify and resolve problems that can occur while you are administering the runtime environment.

Profile-specific log files

There are log files detailing the characteristics and runtime activities of individual profiles. These log files are located within the profile directory for each profile.

There are a number of log files that are created for each profile. Some of these logs describe the parameters used for the creation of the profile. These types of log files generally remain unchanged once the profile is fully configured. Other profile-specific logs are continually updated to capture error, warning, and information messages emitted during run time. Some of these log files are also used to capture a Common Base Event (that might include business object data) that is selected for monitoring.

The table below specifies the different types of profile-specific log files and the locations where you can find them within the product. Within the table, the variable *install_root* represents the installation directory of WebSphere ESB. The variable *profile_root* represents the root location of a profile.

For more information, see Installation directories for the product and profiles.

Table 111. Profile-specific log files updated during runtime

Log	Contents
First failure data capture (ffdc) log and exception files (common to all profile types) are found in these directories: • Linux UNIX **On Linux and UNIX platforms:** *profile_root*/logs/ffdc • Windows **On Windows platforms:** *profile_root*\logs\ffdc	Contains the ffdc log and exception files for individual profiles. There are two types of ffdc logs: a single log file with a compilation of all the errors encountered during the profile runtime, and numerous text files with details such as stack traces and other information. The naming conventions for the different types of profiles are given for both files, as follows: • **Deployment manager profile:** – Log file — *deployment_manager_name*_exception.log. – Text files — *deployment_manager_name_hex_id_date_time*.txt. • **Custom profile:** – Log file(s) — *node_agent_name*_exception.log and *server_name*_exception.log. – Text files — *node_agent_name*(or) *server_name_hex_id_date_time*.txt. • **Stand-alone profile:** – Log file — *server_name*_exception.log. – Text files — *server_name_hex_id_date_time*.txt.

© Copyright IBM Corp. 2007, 2011

Table 111. Profile-specific log files updated during runtime (continued)

Log	Contents
Deployment manager logs (deployment manager profiles only) are found in these directories: • Linux UNIX **On Linux and UNIX platforms:** *profile_root*/logs/*deployment_manager_name* • Windows **On Windows platforms:** *profile_root*\logs*deployment_manager_name*	You will work primarily with four log files in this directory: • `startServer.log` — Contains the system parameters detected on the system and the messages emitted by the deployment manager during the start process • `stopServer.log` — Contains the system parameters detected on the system and the messages emitted when the deployment manager is shut down. • `SystemErr.log` — Contains error and exception messages generated by the deployment manager during runtime. Continually updated while server is running. • `SystemOut.log` — Contains all messages, including error, warning, and information messages generated by the deployment manager during runtime. Continually updated while server is running.
Node agent logs (custom profiles only) are found in these directories: • Linux UNIX **On Linux and UNIX platforms:** *profile_root*/logs/*node_agent_name* • Windows **On Windows platforms:** *profile_root*\logs*node_agent_name*	You will work primarily with four log files in this directory: • `startServer.log` — Contains the system parameters detected on the system and the messages emitted by the node agent during the start process • `stopServer.log` — Contains the system parameters detected on the system and the messages emitted when the node agent is shut down. • `SystemErr.log` — Contains error and exception messages generated by the node agent during runtime. Continually updated while node agent is running. • `SystemOut.log` — Contains all messages, including error, warning, and information messages generated by the node agent during runtime. Continually updated while the node agent is running.

Table 111. Profile-specific log files updated during runtime (continued)

Log	Contents
Server logs (custom and stand-alone profiles only) are found in these directories: • Linux UNIX **On Linux and UNIX platforms:** *profile_root*/logs/*server_name* • Windows **On Windows platforms:** *profile_root*\logs*server_name*	You will work primarily with four log files in this directory: • `startServer.log` — Contains the system parameters detected on the system and the messages emitted by the server during the start process • `stopServer.log` — Contains the system parameters detected on the system and the messages emitted when the server is shut down. • `SystemErr.log` — Contains error and exception messages generated by the server during runtime. Continually updated while server is running. • `SystemOut.log` — Contains all messages, including error, warning, and information messages generated by the server during runtime. Also contains any events being monitoring that are emitted from the Common Event Infrastructure (CEI), in Common Base Event format. These events may also include the level of business object data (FINE, FINER, or FINEST) that is specified for the monitor. Continually updated while the server is running.
Node federation log files are found in these directories (only applies to non-deployment manager profiles): • Linux UNIX **On Linux and UNIX platforms:** *profile_root*/logs • Windows **On Windows platforms:** *profile_root*\logs	Two log files are generated when you attempt to federate a custom, augmented, or stand-alone profile to a deployment manager: • `addNode.log` — contains the pertinent server environment information and messages generated when you attempt to federate the profile. • `isFederated.log` — lists the commands used by the deployment manager to federate the profile.
The location of the Integrated Solutions Console application deployment log file is listed here (only for deployment manager and stand-alone profiles): • Linux UNIX **On Linux and UNIX platforms:** *profile_root*/logs/iscinstall.log • Windows **On Windows platforms:** *profile_root*\logs\iscinstall.log	The iscinstall.log file contains information regarding the deployment of the administrative console application in a deployment manager or stand-alone profile.
The location of the Installation Verification Tool log file is listed here (only for deployment manager and stand-alone profiles): • Linux UNIX **On Linux and UNIX platforms:** *profile_root*/logs/ivtClient.log • Windows **On Windows platforms:** *profile_root*\logs\ivtClient.log	This log file contains the output generated by the Installation Verification Tool. You can start this program from the First Steps console after you create a deployment manager or stand-alone profile. The log contains basic configuration information and the messages that are displayed when you run the tool.

Table 111. Profile-specific log files updated during runtime (continued)

Log	Contents
The location of the log file detailing the commands generated for a profile creation is listed here: • Linux UNIX **On Linux and UNIX platforms:** `profile_root/logs/updateserverpolicy.log` • Windows **On Windows platforms:** `profile_root\logs\updateserverpolicy.log`	This file contains the sequence of commands used by the product to set server environment variables and create a profile. All profile types will contain this file.

Related concepts:

Chapter 17, "WebSphere ESB log files," on page 1035
WebSphere ESB offers a comprehensive set of log files to help you identify and resolve problems during installation, configuration, and run time.

Related information:

Installation and profile creation log files
Various log files are created during installation and uninstallation of WebSphere ESB and during profile creation, augmentation, and deletion. Consult the applicable logs if problems occur during these procedures.

Resolving browser display problems with administration tools

If you use Microsoft Internet Explorer V8 to access Process Center console, Process Admin Console, Performance Admin console, or IBM Process Portal, you can have trouble viewing the page contents in the browser.

About this task

To ensure all of the pages in these administration tools load correctly, run Internet Explorer V8 in Compatibility View. See the Internet Explorer documentation for details on enabling Compatibility View.

Note: This limitation does not apply to Business Space or the WebSphere Application Server administrative console.

Troubleshooting the failed event manager

This topic discusses problems that you can encounter while using the failed event manager.

Select the problem you are experiencing from the table below:

Problem	Refer to the following
I am having trouble entering values in the Search page's **By Date** tab	"Values in the By Date and From Date field automatically change to default if entered incorrectly" on page 580
I am having trouble deleting expired events	"Using the Delete Expired Events function appears to suspend the failed event manager" on page 581
I am having trouble with failed events not being created	"Failed events are not being created" on page 581
I am having trouble retrieving or deleting large numbers of failed events	"The server fails when retrieving or deleting large numbers of failed events" on page 581

Values in the By Date and From Date field automatically change to default if entered incorrectly

The Search page's **From Date** and **To Date** fields require correctly formatted locale-dependent values. Any inconsistency in the value's format (for example, including four digits in the year instead of 2, or omitting the time) will cause the failed event manager to issue the following warning and substitute a default value in the field:

```
CWMAN0017E: The date entered could not be parsed correctly:
your_incorrectly_formatted_date. Date: default_date is being used.
```

The default value of the **From Date** field is defined as `January 1, 1970, 00:00:00 GMT`.

Important: The actual default value shown in your failed event manager implementation will vary depending on your locale and time zone. For example, the From Date field defaults to `12/31/69 7:00 PM` for a workstation with an en_US locale in the Eastern Standard Time (EST) time zone.
The default value for the **To Date** field is always the current date and time, formatted for your locale and time zone.

To avoid this problem, always enter your dates and times carefully, following the example provided above each field.

Using the Delete Expired Events function appears to suspend the failed event manager

If you use the Delete Expired Events button in situations where there are many failed events in the current search results, or where those events contain a large amount of business data, the failed event manager can appear to be suspended indefinitely.

In this situation, the failed event manager is not suspended: it is working through the large data set, and will refresh the results set as soon as the command completes.

Failed events are not being created

If the Recovery subsystem is not creating failed events, go through the following checklist of potential causes:

- Ensure that the wpsFEMgr application is running. If necessary, restart it.
- Ensure that the failed event manager's database has been created, and that the connection has been tested.
- Ensure that the necessary failed event destination has been created on the SCA system bus. There should be one failed event destination for each deployment target.
- Ensure that the Quality of Service (QoS) **Reliability** qualifier has been set to `Assured` for any Service Component Architecture (SCA) implementation, interface, or partner reference that participates in events you want the Recovery service to handle.

The server fails when retrieving or deleting large numbers of failed events

The server can fail if you use the failed event manager to retrieve or delete a large number of failed events at once. To prevent this problem, be sure to check the total failed event count on the main page of the failed event manager before performing search or delete actions. If you have a large number of events, do not try to retrieve or delete all of them at once. Instead, use the **Search failed events** option to return a subset of failed events that match specific criteria. You can then delete all of the events in the filtered result set without causing a server failure.

Note: There are several factors in your environment that can affect the number of failed events the server can return or delete in a single request, including other processes running on the same machine as the server and the amount of available memory.

Troubleshooting store-and-forward processing

This topic discusses problems that you can encounter with store-and-forward processing.

Select the problem you are experiencing from the table below:

Problem	Refer to the following
I am having problems setting the store-and-forward qualifier	"Store-and-forward qualifier processing only works on asynchronous interfaces"
Qualifying runtime exceptions are occurring, but events are not getting stored	"Store is not activated by qualifying runtime exceptions" on page 1065
Messages are still being processed even though the Store and Forward widget shows the state is set to Store (Network deployment environment)	"In a network deployment environment, messages are being processed even though the store-and-forward state is set to Store" on page 1065
The Store and Forward widget shows the state is set to Forward, but messages are not being processed by all members of the cluster. (Network deployment environment)	"In a network deployment environment, messages are not getting processed by all members of the cluster even though the store-and-forward state is set to Forward" on page 1065

Store-and-forward qualifier processing only works on asynchronous interfaces

The store-and-forward qualifier must be specified on an asynchronous interface. The store cannot be activated if the interface is called synchronously.

Here are some guidelines (with respect to components) to help you determine if the interface is being called synchronously or asynchronously.

- Mediation flows invoke imports based on the preferred interaction style set on the import's interface. Look at the interaction style set on the import's interface to see whether it is synchronous or asynchronous.

  Note: You can find this setting on the interface's detail tab.
- POJO components invoke components based on the code that is written in the component. Look at the code written in the component to see whether it is synchronous or asynchronous.

Store is not activated by qualifying runtime exceptions

If the store is not being activated by qualifying runtime exceptions, check the following.

- The exception specification in the store-and-forward qualifier matches the exception that occurs at runtime. If the exception specification does not match, storing does not activate.
- The user code in the path is not catching the exception and wrapping it. Or, it is converting it into a different exception. The exception received by the store-and-forward function can be viewed in the exception details for the failed event.
- The destination component for a failed event has a store-and-forward qualifier set on it. Storing is activated once a failed event is generated. If a failed event is generated for a component that is upstream from the component that has a store-and-forward qualifier set on it, then the store-and-forward component is being invoked synchronously and not asynchronously. If a failed event is generated for a component that is downstream from the store-and-forward qualifier component, rather than the component with the store-and-forward qualifier set on it, then there is an asynchronous invocation closer to the failure and the store-and-forward qualifier should be moved to that component.

In a network deployment environment, messages are being processed even though the store-and-forward state is set to Store

Messages might continue to be processed by some members of a cluster, despite the state being set to Store, if the state is not set to Store for each member of the cluster. To fix this problem, confirm that the state is set to Store for each member of the cluster in the Store and Forward widget. If any members of the cluster are set to Forward, change them to Store.

This might also happen if one of the members of the cluster is forced to restart. Since the Store state is not persistent, it reverts to the Forward state at restart. To fix this problem, change the state to Store for the module in the Store and Forward widget.

Note: When the service becomes available again, you should not set the state to Store immediately if you want new events to be processed. If you set the state to Store before new events have the chance to be processed, they will be stored in the queue.

In a network deployment environment, messages are not getting processed by all members of the cluster even though the store-and-forward state is set to Forward

Messages might continue to be stored by some members of a cluster, despite the state being set to Forward, if the store-and-forward state is not set to Forward for each member of the cluster. To fix this problem, confirm that the state is set to Forward for the module in the Store and Forward widget. If any members of the cluster are set to Store, change them to Forward.

Note:

Chapter 23. Troubleshooting WebSphere Application Server

Because WebSphere ESB is built on IBM WebSphere Application Server, the function that you are having problems with might be provided by the underlying WebSphere Application Server. You might want to consult troubleshooting information in the WebSphere Application Server documentation.

WebSphere ESB is built on WebSphere Application Server Network Deployment, version 7.0.

For more information about troubleshooting in WebSphere Application Server, see the topic "Troubleshooting and support" in the WebSphere Application Server Information Center.

Related information:

Troubleshooting and support

© Copyright IBM Corp. 2007, 2011

Chapter 24. Tools for troubleshooting your applications

WebSphere ESB and Integration Designer include several tools you can use to troubleshoot your applications that you develop and deploy on the server.

During development of your applications, you can use debugging tools in Integration Designer. You can implement runtime troubleshooting capabilities into your applications using logging, tracing, and service component event monitoring.

Debugging applications in IBM Integration Designer

To debug applications that are running on WebSphere ESB, use your application development tool, such as IBM Integration Designer.

About this task

For more information about debugging applications, see “Component debugging”.

Related information:

Component debugging

Using logging, tracing, and monitoring in your applications

Designers and developers of applications that run on WebSphere ESB can use capabilities such as monitoring and logging that add troubleshooting features to applications.

About this task

WebSphere ESB is built on IBM WebSphere Application Server, Network Deployment. For more information, see the topic “Adding logging and tracing to your application” in the WebSphere Application Server Information Center.

To use logging, tracing, and monitoring with your applications, perform the steps in the Procedure section.

Procedure

- You can set up service component event monitoring for applications running on WebSphere ESB. For more information, see the “Monitoring service component events” topic link in the Related Topics section at the bottom of this page.
- You can add logging and tracing to your applications using WebSphere Application Server.

Related information:

Adding logging and tracing to your application

Monitoring events

Collecting problem reports using IBM Support Assistant Lite

Using IBM Support Assistant Lite (ISA Lite), which is preinstalled in IBM Business Process Manager, you can search for information, investigate problems, or submit a problem report to IBM.

© Copyright IBM Corp. 2007, 2011

About this task

Use IBM Support Assistant Lite for Business Process Manager to identify problems, collect information about them, diagnose them, and solve them. For more information, see the description of ISA Lite on the IBM Support Portal. The Support Portal also offers tips for using the tool effectively. The IBM Education Assistant site provides a tutorial about ISA Lite, although you can ignore the instructions about downloading and installing, because the utility is preinstalled.

Procedure

To run the ISA Lite collectors and generate problem reports, complete these steps. You might have other versions of IBM Support Assistant on your system, but if you run the commands recommended below, you will collect the appropriate information for this product.

1. In a command window, navigate to the following directory: *WAS install root*/ISALiteBPM.
2. For Windows systems, run the **runISALite.bat** command. For Unix or Linux systems, run the **runISALite.sh** command.
3. From the **Data Collection Options** menu, select the category of problem that you want to investigate. For an explanation of the categories and conditions under which each should be used, see "IBM Support Assistant Lite collectors" in the related links.
4. Specify where you want the collector's output file to be placed.
5. Follow the prompts to provide additional information that the collector will use to gather appropriate data for the trace files. The collector takes about 15 to 20 minutes to run. The message in the Progress window tells you when the data collection has completed successfully.
6. At the end of the collection, you can choose to send the collection to an IBM Support FTP site.

Related concepts:

"Problem resolution using IBM Support Assistant Lite collectors"
You can use IBM Support Assistant Lite to collect information about a number of specific problems. Choose the collector that is appropriate for the troubleshooting issue that you are trying to solve.

Related tasks:

Chapter 29, "Contacting IBM Software Support," on page 1131
IBM Software Support provides assistance with product defects.

Chapter 28, "Getting fixes," on page 1129
A product fix might be available to resolve your problem.

Related information:

IBM Support Assistant

Problem resolution using IBM Support Assistant Lite collectors

You can use IBM Support Assistant Lite to collect information about a number of specific problems. Choose the collector that is appropriate for the troubleshooting issue that you are trying to solve.

Generic problem collector
: Use this collector only if the product installed successfully, no server processes are running, and a server profile exists. The generic problem collector gathers configuration information and logs from a profile.

Log and trace collector
Use this collector only if the product installed successfully. This utility collects log and trace files for a profile. It is faster than the generic problem collector.

Migration problem
Use this collector if a problem occurs during migration. Before you can use this collector, the product must have installed successfully, a server profile must exist, and no server processes can be running.

Installation problem
Use this collector if a problem occurs during installation. Before you can use this collector, IBM Installation Manager must have installed successfully.

Profile wizard problem
Use this collector if you encounter a problem when creating or augmenting a profile. Before you can use this collector, IBM Business Process Manager must have installed successfully.

Reproduce problem with tracing level set

Use this collector to set trace levels to reproduce a problem. The utility starts and stops the server.

Important: Be aware that this collector makes changes to the server tracing and tries to restore the original settings. Do not run this collector if you want to avoid these actions.

Before you can use this collector, IBM Business Process Manager must have installed successfully. A profile must exist and you must be able to start it successfully.

Related tasks:

"Collecting problem reports using IBM Support Assistant Lite" on page 1069
Using IBM Support Assistant Lite (ISA Lite), which is preinstalled in IBM Business Process Manager, you can search for information, investigate problems, or submit a problem report to IBM.

Troubleshooting Service Component Architecture processing and call chains

Cross-Component Trace identifies whether a Service Component Architecture (SCA) operation completed successfully. It allows you to identify systemout.log or trace.log data that is associated with WebSphere ESB and WebSphere Enterprise Service Bus modules and components. The log records associated with the WebSphere Enterprise Service Bus applications hold information about errors or events that occurred during processing and can be used for problem determination using IBM Integration Designer.

Events that can be captured include:

- Errors that occur during processing because of corrupted data
- Errors when resources are not available, or are failing
- Interpretation of code paths

You can access the Cross-Component Trace page from the administrative console by clicking **Troubleshooting** > **Cross-Component Trace**.

Handling and deleting collected data

Consider the following with regard to handling and deleting data collected by Cross-Component Trace:

- SCA call chain information is added to the systemout.log and trace.log files and is purged as those files are purged.
- Data snapshots capture the input and output data of call chains.

  The input and output data is captured as files in the `logs\XCT` directory. To display this data, IBM Integration Designer needs access to the systemout.log files and the `logs\XCT` directory. If IBM Integration Designer is not available on the server, copying the logs directory and placing it on a machine (so that it can be accessed by IBM Integration Designer) preserves the file structure so that IBM Integration Designer can make use of the log files and the data snapshot files.

  Note: IBM Integration Designer can use the data snapshot files where they are (without moving them) if it can access the files in the logs directory. If you need to move files, it is safest to move the entire logs directory. By moving the entire logs directory, you get the XCT, first failure data capture (FFDC) files, and the systemout.log and trace.log files.

  Data snapshot files are written to server-specific subdirectories using the following directory structure:

  ```
  logs\
       server
       ffdc
       xct\
          server-specific_dir\
                                2009-0-25-11
                                2009-0-26-12
                                2009-0-26-14
  ```

  Where `server-specific_dir` name is derived from the name of the server. For example, **server1** is the default server name for a stand-alone installation.
- Data snapshot files in `logs\XCT\server` are referenced from the systemout.log and trace.log files that were created at the same time by the server. When the old systemout.log and trace.log files are deleted, the associated Cross-Component Trace data snapshot files in `logs\XCT\server` can also be deleted.

  You can use the timestamps in the systemout.log and trace.log files to identify and determine what data snapshot files to delete. It is safe to delete all the data snapshot files for a server that are older than the oldest date in the systemout.log and trace.log files. Preferably, you should use the **Delete data snapshot files** function from the administrative console when data snapshot files are no longer needed. For detailed information on the ways that you can delete data snapshot files, see “Deleting data snapshot files”.
- Do not save or add files to the `logs\XCT` directory. Do not copy or create new directories into the `logs\XCT` directory.

  WebSphere ESB manages the content of the `logs\XCT` directory and deletes items that are no longer needed. WebSphere ESB might consider unrecognized files or directories as unnecessary and delete them. If you want to save a copy of the data snapshot files, copy the data to another directory outside of the `logs\XCT` directory.

Cross-Component Trace settings and call chain processing

The information in this section describes the effect that Cross-Component Trace configuration settings have on call-chain processing. It also includes a description of various Cross-Component Trace configurations and explains the call chain events that result from the configurations.

The following list includes general rules on call chain processing and Cross-Component Trace configuration decisions:

- If Cross-Component Trace is turned off for a server, then no Cross-Component Trace records are written to that server's logs.
- Cross-Component Trace configuration settings for a particular server, *affect that server only*.

  For example, if Server A has **Trace all = Yes** and Server B has **Trace all = No**, Cross-Component call chains are in the logs for Server A only. Similarly, this rule applies to setting the data snapshot feature. If **Enable data snapshot = Yes** on Server A and **Enable data snapshot = No** on Server B, then only Server A will have data snapshot files in its logs directory.
- Application-specific Cross-Component Trace data flows between servers that have the **Enable Cross-Component Trace = Yes**.

  For example, if both Server A and Server B have **Enable Cross-Component Trace = Yes** and Server A has enabled Cross-Component Trace for a specific SCA module, the calls made from the Cross-Component Trace-enabled module on Server A (to applications or services on Server B), will result in Server A having call chains for all of activity related to the Cross-Component Trace-enabled module. Server B would also have call chains, but only for those calls that came from the Cross-Component Trace-enabled module on Server A. The logs of the two servers can be combined to reveal the entire call chain activity.
- To create a Cross-Component Trace for long-running BPEL process instances, you must select **Enable Cross-Component Trace** and **Trace all**, or enable Cross-Component Trace for the desired SCA module before the BPEL process instance is created.

  For more information see “Enabling Cross-Component Trace for BPEL long-running instances” in the Cross-Component Trace overview.

The following illustration is of two servers (Server A and Server B), both with Cross-Component Trace enabled. Server A has the **Trace all** value set to “Yes”, while Server B has **Trace all** set to “No”.

Figure 317. A remote messaging and remote support topology

For the Cross-Component Trace configuration scenario illustrated in Figure 317, call chain events would result on Server A, but not on Server B.

The following illustration is of two servers (Server A and Server B), both with Cross-Component Trace enabled. Server A has the **Trace all** value set to No and it includes Module A as a module on which to enable Cross-Component Trace. Server B has **Trace all** set to No and has no SCA modules selected for Cross-Component Trace.

Figure 318. A remote messaging and remote support topology

For the Cross-Component Trace configuration scenario illustrated in Figure 318, call chain events would result on Server A. Trace activity for all Module A operations are written to the log on Server A. Any calls made from Module A to applications or services on Server B, results in call chains. The call chains on Server B would

only pertain to those calls that came from Module A (because that module is configured for Cross-Component Trace).

Managing failed events in WebSphere ESB

When an event fails in WebSphere ESB, it is stored in a database in the failed events manager. You can search for a failed event, and then view, modify, resubmit, or delete it.

To view, modify, resubmit, or delete any failed event, as described in the following sections, the first step is to click **Integration Applications** > **Failed Event Manager**.

Failed SCA events

A Service Component Architecture (SCA) event is a request or response that is received by a service application. It might come from an external source, such as an inbound application adapter, or an external invocation to a web service. The event consists of a reference to the business logic that it wants to operate and its data, which is stored in a Service Data Object. When an event is received, it is processed by the appropriate application business logic.

A single thread of execution might branch off into multiple branches. The individual branches are linked to the main invoking event by the same session.

If the business logic in a branch cannot run due to system failure, component failure, or component unavailability, a failed event is created. If multiple branches fail, a failed event is created for each branch. The Recovery service handles the following types of failed SCA events:

- Event failures that occur during an asynchronous invocation of an SCA operation
- Event failures that are caused by a runtime exception, for example, any exception that is not declared in the methods used by the business logic

The Recovery service does not handle failures from synchronous invocations.

Failed SCA events typically have source and destination information associated with them. The source and destination are based on the failure point (the location where the invocation fails), regardless of the type of interaction. Consider the following example, where Component A is asynchronously invoking Component B. The request message is sent from A to B, and the response callback message is sent from B to A.

- If the exception occurs during the initial request, Component A is the source and Component B is the destination for the failed event manager.
- If the exception occurs during the response, Component B is the source and Component A is the destination for the failed event manager.

This pattern is true for all asynchronous invocations.

The Recovery service sends failed SCA asynchronous interactions to failed event destinations that have been created on the SCA system bus (SCA.SYSTEM.*cell_name*.Bus). The data for failed events is stored in the failed event database (by default, WPCRSDB). You can find the data in the failed event manager.

Failed WebSphere MQ events

A WebSphere MQ event might fail if there is a problem such as a data-handling exception in the WebSphere MQ binding export or import used by an SCA module.

You can use the recovery binding property to enable or disable recovery for each WebSphere MQ binding at authoring time. You can set the `recoveryMode` property to one of the following settings:

`bindingManaged`	Allow binding to manage recovery for failed messages
`unmanaged`	Rely on transport-specific recovery for failed messages

Recovery for WebSphere MQ bindings is enabled by default. When it is enabled, WebSphere MQ failed events are created in the following situations:

- The function selector fails
- The fault selector fails
- The fault selector returns the `RuntimeException` fault type
- The fault handler fails
- The data binding or data handler fails after another attempt in WebSphere MQ

In addition, a failed SCA event is created when the `ServiceRuntimeException` exception is thrown in a WebSphere MQ binding target component after another attempt in WebSphere MQ.

These failures might occur during inbound or outbound communication. During outbound communication, MQImport sends a request message and receives the response message. A failed event is generated if the WebSphere MQ import binding detects a problem while it is processing the service response. During inbound communication, the following sequence of events occurs:

1. MQExport receives the request message.
2. MQExport starts the SCA component.
3. The SCA component returns a response to MQExport.
4. MQExport sends a response message.

A failed event is generated if the WebSphere MQ export binding detects a problem while it is processing the service request.

The Recovery service captures the WebSphere MQ message and stores it in the failed event database. It also captures and stores the module name, component name, operation name, failure time, exception detail, and WebSphere MQ properties of the failed event. You can use the failed event manager to manage failed WebSphere MQ events, or you can use a custom program.

To disable recovery, set the `recoveryMode` property to `unmanaged`.

Important: If the `recoveryMode` property is missing for earlier versions of applications, the recovery capability is regarded as enabled.
When recovery is disabled, a failed message is rolled back to its original destination and tried again. The system does not create a failed event.

Failed JMS events

The Java Message Service (JMS) binding type and configuration determine whether a failed event is generated and sent to the failed event manager.

You can use the recovery binding property to enable or disable recovery for each JMS binding at authoring time. You can set the `recoveryMode` property to one of the following settings:

`bindingManaged`	Allow binding to manage recovery for failed messages
`unmanaged`	Rely on transport-specific recovery for failed messages

Recovery for JMS bindings is enabled by default. When it is enabled, JMS failed events are created in the following situations:

- The function selector fails
- The fault selector fails
- The fault selector returns the `RuntimeException` fault type
- The fault handler fails
- The data binding or data handler fails after one repetition in JMS

In addition, a failed SCA event is created when the `ServiceRuntimeException` exception is thrown in a JMS binding target component after another attempt in JMS.

These failures might occur during inbound or outbound communication. During outbound communication, JMSImport sends a request message and receives the response message. A failed event is generated if the JMS import binding detects a problem while it is processing the service response. During inbound communication, the following sequence of events occurs:

1. JMSExport receives the request message.
2. JMSExport starts the SCA component.
3. The SCA component returns a response to JMSExport.
4. JMSExport sends a response message.

A failed event is generated if the JMS export binding detects a problem while it is processing the service request.

The Recovery service captures the JMS message and stores it in a Recovery table in the Common database. It also captures and stores the module name, component name, operation name, failure time, exception detail, and JMS properties of the failed event. You can use the failed event manager to manage failed JMS events, or you can use a custom program.

To disable recovery, set the `recoveryMode` property to `unmanaged`.

Important: If the `recoveryMode` property is missing for earlier versions of applications, the recovery capability is regarded as enabled.
When recovery is disabled, a failed message is rolled back to its original destination and tried again. The system does not create a failed event.

Tip: For service runtime exceptions that are generated when a requested service is unavailable, you can use the Store and Forward feature to prevent further failures.

You specify a store-and-forward qualifier when you configure a component to be started asynchronously. When a runtime error is generated by that component, subsequent events (in this case, asynchronous requests) are prevented from reaching the component. For more information, see “Preventing failures when a service is unavailable”.

Related concepts:

“Preventing multiple failures when a service is unavailable” on page 582
If your component calls a service, and that service is unavailable, a runtime error occurs. Events associated with the error are treated, by default, as failed events. Using the store-and-forward feature, however, you can drastically reduce the number of failed events that are generated by the system.

Security considerations for recovery

If you have enabled security for your WebSphere ESB applications and environment, it is important to understand how role-based access and user identity affect the Recovery subsystem.

Role-based access for the failed event manager
The failed event manager uses role-based access control for the failed event data and tasks. Only the administrator and operator roles are authorized to perform tasks within the failed event manager. Users logged in as either administrator or operator can view all data associated with failed events and can perform all tasks.

Event identity and user permissions
A failed event encapsulates information about the user who originated the request. If a failed event is resubmitted, its identity information is updated to reflect the user who resubmitted the event. Because different users logged in as administrator or operator can resubmit events, these users must be given permissions to the downstream components required to process the event.

For more information about implementing security, see Securing applications and their environment.

Finding failed events in WebSphere ESB

You can use the failed event manager to find failed events in WebSphere ESB. You can search for all failed events, or for a specific subset of events, on all the servers within the cell.

Before you begin

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

About this task

This topic describes how to find all failed events in a cell. This default query returns all failed events for Service Component Architecture (SCA) and Java Message Service (JMS). If Business Process Choreographer is installed, the query also returns failed, terminated, and stopped Business Process Choreographer events. For more information about different types of failed events, see "Managing failed events in WebSphere ESB".

Procedure

To retrieve a complete list of failed events, complete the following steps:

1. Ensure that the administrative console is running.
2. Click **Integration Applications** > **Failed Event Manager**.
3. In the **Failed events on this server** box, click **Get all failed events**.

Results

The Search Results page opens with a list of all the WebSphere ESB failed events in the cell.

What to do next

You can now view (and in some cases, modify) data in a failed event, resubmit it, or delete it.

Searching for events by criteria

Use the failed event manager Search page to find only those events that meet specified criteria. You can search by failed event type and by criteria such as failure time, event destination or source, exception or business object type, session ID or event sequencing qualifier.

Before you begin

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

About this task

To search for a specific subset of failed events on the server, perform the following steps.

Procedure

1. Ensure the administrative console is running.
2. Click **Integration Applications** > **Failed Event Manager** to enter the failed event manager.
3. From the **Failed events on this server** box, click **Search failed events**.
4. From the **Event type** box on the Search failed events page, select one or more types of events to search for:
   - SCA
   - JMS
   - WebSphere MQ
5. Optional: Specify any additional search criteria. The following table describes the available options. If you specify multiple criteria, the AND operator is used during the query; the failed event manager returns only events that meet all of the criteria.

Table 112. Search criteria

Search criteria	Field or fields to use	Supported event types	Usage notes
The module, component, or method the event was en route to when it failed.	**Module** **Component** **Operation**	SCA JMS WebSphere MQ	Use one or more of these fields to search for failed events associated with a specific module, component, or method.
The time period during which the event failed	**From date** **To date**	SCA JMS WebSphere MQ	Formats for date and time are locale-specific. An example is provided with each field. If the value you provide is not formatted correctly, the failed event manager displays a warning and substitutes the default value for that field. The time is always local to the server. It is not updated to reflect the local time of the individual workstations running the administrative console.
The session in which the event failed	**Session ID**	SCA	None
The module or component from which the event originated	**Source module** **Source component**	SCA	Use one or both of these fields to find only those failed events that originated from a specific source module or component. The failed event manager determines the source based on the point of failure, regardless of interaction type.
The type of business object in the failed event	**Business object type**	SCA	None
Whether the event had the event sequencing qualifier specified	**Event sequencing qualifier**	SCA	None
Whether the event caused the store to be started	**Store and forward qualifier**	SCA	None
Whether the event was caused because a failure response could not be sent to Business Process Choreographer	**Process response qualifier**	SCA	None
The exception thrown when the event failed	**Exception text**	SCA	Specify all or part of the exception text in the field to find all events associated with that exception.

For detailed information about each field and the values it accepts, see the online help for the failed event manager Search page.

6. Click **OK** to begin the search.

What to do next

You can now view (and in some cases, modify) data in a failed event, resubmit it, or delete it.

Working with data in failed events

Each failed event has data associated with it; often, that data can be edited before an event is resubmitted. There are two basic types of data for a failed event: data about the event, and business data.

Data about the failed event

All failed events have the following data:

- The event ID, type, and status
- The time the event failed
- The deployment target associated with the event

In addition, SCA, JMS, WebSphere MQ, Business Process Choreographer, and Business Flow Manager hold queue events have data specific to the event type:

- SCA events
  - The session ID
  - The type of service invocation used between SCA components
  - The names of the module and component from which the event originated (the source).
  - The names of the destination module, component and method for the event
  - Whether an event sequencing qualifier has been declared for this event
  - The destination module where the event has been or will be resubmitted
  - The correlation ID, if one exists
  - The exception thrown when the event failed
  - The expiration date for resubmitted events (this data can be edited)
  - The trace control set for the event (this data can be edited)
- JMS events:
  - The type of service invocation used
  - The names of the destination module, component and method for the event
  - The exception thrown when the event failed
  - The destination module where the event has been or will be resubmitted
  - The correlation ID, if one exists
  - The expiration date for resubmitted events (this data can be edited)
  - The JMS-specific properties associated with the failed event:
    - The message type and priority
    - The JMS destination
    - The delivery mode
    - Redelivery data, including the redelivered count and redelivered indicator (`true` or `false`)
    - The destination replies are sent to for request-response or two-way interactions
- WebSphere MQ events:
  - The type of service invocation used

- The names of the destination module, component and method for the event
- The exception thrown when the event failed
- The destination module where the event has been or will be resubmitted
- The correlation ID, if one exists
- The expiration date for resubmitted events (this data can be edited)
- The WebSphere MQ-specific properties associated with the failed event:
  - The message type, format, and priority
  - The WebSphere MQ destination
  - The delivery mode
  - Redelivery data, including the redelivered count and redelivered indicator (`true` or `false`)
  - The reply-to queue and queue manager

Business data

SCA failed events typically include business data. Business data can be encapsulated in a business object, or it can be simple data that is not part of a business object. Business data for SCA failed events can be edited with the business data editor available in the failed event manager.

Browsing data in failed events

Use the failed event manager to view failed event data and any business data associated with the event.

Before you begin

If administrative security is enabled, you must be logged as administrator or operator to perform this task.

About this task

To browse failed event data, use the following procedure.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the Search Results page of the failed event manager, click the ID (found in the Event ID column) of the failed event whose data you want to browse.

   The Failed Event Details page opens and displays all of the information about the event.
3. If your failed event has business data, you can browse it by clicking **Edit business data**.

   The Business Data Editor collection page opens, displaying the business data associated with the failed event. Each parameter name in the hierarchy is a link. If the parameter is a simple data type, clicking its name will open up a form so you can edit the parameter's value. If the parameter is a complex data type, clicking its name will expand the hierarchy further.

Editing trace or expiration data in a failed SCA event

The Failed Event Details page enables you to set or modify values for the trace control and expiration date associated with a failed event.

Before you begin

You must be logged in as administrator or operator to perform this task.

About this task

Important: Any edits you make to the trace or expiration data are only saved locally until you resubmit the event. If you perform any other action before resubmitting the event, all edits are lost.

Failed Service Component Architecture (SCA) events can be resubmitted with trace to help you monitor the event processing. Tracing can be set for a service or a component, and it can be sent to a log or to the Common Event Infrastructure (CEI) server. When you view the failed event data on the Failed Event Details page, the default trace value `SCA.LOG.INFO;COMP.LOG.INFO` is shown for the event. If you resubmit the event with this default setting, no trace occurs when the session calls an SCA service or executes a component.

Some failed SCA events also have an expiration. If a user has specified an expiration with the asynchronous call that sends the event, that data persists even if the event fails, and the expiration time appears in the **Resubmit Expiration Time** field on the Failed Event Details page. Expired failed events cannot be resubmitted successfully. To prevent a second failure, you can edit the expiration date for the event to ensure that it is not expired when it is resubmitted.

To edit trace or expiration data in a failed event, use the following procedure.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the failed event manager's Search Results page, click the ID (found in the Event ID column) of the failed event whose data you want to edit.

   The Failed Event Details page opens.
3. If the event has an expiration date that causes it to expire before it is resubmitted, edit the expiration in the **Resubmit expiration time** field.

   The expiration time shown is local to the server. The value for this field must be formatted according to your specified locale. An example of the correct format for your locale is provided above the field.
4. If you want to enable tracing for the failed event, specify a new value in the **Trace Control** field. For detailed information about trace values, see the Monitoring topics in the WebSphere Business Process Management Information Center.
5. Do one of the following:
   - If the edited data is correct and you want to resubmit the event, click **Resubmit** to make the changes at a server level.
   - If you want to remove the changes you made, click **Undo local changes**.

   The edited failed event is resubmitted for processing and is removed from the failed event manager.

Related tasks:

"Finding failed events in WebSphere ESB" on page 566
You can use the failed event manager to find failed events in WebSphere ESB. You can search for all failed events, or for a specific subset of events, on all the servers within the cell.

Editing business data in a failed SCA event

Business data can be encapsulated into a business object, or it can be simple data that is not part of a business object. A failed event can have both simple data and a business object associated with it. Use the business data editor to edit the business data associated with a failed event before you resubmit it.

Before you begin

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

About this task

For each failed event, the editor displays the associated business data in a hierarchical format; the navigation tree is updated as you navigate through the parameters to give you a clear picture of where you are in the hierarchy.

You can edit only simple data types (for example, String, Long, Integer, Date, Boolean). If a data type is complex (for example, an array or a business object), you must navigate through the business data hierarchy until you reach the simple data types that make up the array or business object. Complex data is denoted by an ellipsis (`...`) in the Parameter Value column.

Important: Any edits you make to business data are saved locally. Changes are not made to the corresponding business data in the server until you resubmit the failed event.

To edit business data associated with a failed Service Component Architecture (SCA) event, use the following procedure.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the failed event manager's Search Results page, click the ID (found in the Event ID column) of the failed event that has data you want to edit.
3. From the failed event detail page, click **Edit business data** to access the Business Data Editor collection page.

   This page displays a hierarchical view of all the data associated with the failed event.
4. Navigate through the business data hierarchy by clicking the name of each parameter (these appear as links in the Parameter Name column). When you have located the parameter with values you want to edit, click its name.

   If the parameter has an editable value, the Business Data Editor page opens.
5. In the **Parameter value** field, specify the new value for the parameter.
6. Click **OK**.

   The change is saved locally and you are returned to the Business Data Editor collection page.

7. If you want to remove the changes you made, click **Undo local business data changes**.

   All the edits are removed and the business data is returned to its original state.
8. If the edited business data is correct, click **Resubmit** to make changes at a server level.

   The edited failed event is resubmitted for processing and is removed from the failed event manager.

Resubmitting failed events in WebSphere ESB

You can resubmit a failed event in WebSphere ESB from the failed event manager. You can resubmit an event without changes, or, in some cases, you can edit the business data parameters before you resubmit it.

When a failed event is resubmitted, the processing resumes only for the failed branch and not for the entire event.

Important: If one of the following conditions occurs while an application is running in a unit-test environment or a production environment, you cannot resubmit the failed event:

- The application is stopped on the server
- The input and output parameters are modified
- The schema for the input and output parameters is modified

You can trace resubmitted SCA events to monitor the processing of the event. You can set tracing for a service or a component, and you can send the output to a log or to the Common Event Infrastructure (CEI) server.

You can also use the unique ID of the event to track its success or failure. If a resubmitted event fails again, it is returned to the failed event manager with its original event ID and an updated failure time.

Resubmitting an unchanged failed event

You can resubmit one or more unchanged failed events to be processed again. Processing resumes only for the failed branch, not for the entire event.

About this task

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the Search Results page, select the check box next to each failed event you want to resubmit.
3. Click **Resubmit**.

Results

Each selected event is resubmitted for processing and is removed from the failed event manager.

Resubmitting a failed SCA event with trace

You can monitor the resubmission of a failed Service Component Architecture (SCA) event to determine whether it now succeeds. The failed event manager provides optional tracing for all failed events.

About this task

Tracing can be set for a service or a component, and it can be output to a log or to the Common Event Infrastructure (CEI) server. For detailed information about setting and viewing trace, see Monitoring.

If administrative security is enabled, you must be logged in as administrator or operator to perform this task.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the Search Results page, select the check box next to each failed event you want to resubmit.
3. Click **Resubmit with trace**.
4. From the Resubmit with Trace page, specify the level of trace you want to use in the **Trace control** field.

   By default, the value is SCA.LOG.INFO;COMP.LOG.INFO. With this setting, no trace occurs when the session calls an SCA service or executes a component.
5. Click **OK** to resubmit the failed event and return to the Search Results page.

What to do next

To view the trace log for a resubmitted event, open the corresponding component logger or use the CEI log viewer.

Resubmitting failed Business Process Choreographer responses

When a failure response cannot be delivered to the requesting business process due to an infrastructure problem, an event is stored in the failed event database. These type of events will have a process response qualifier specified on them. You can resubmit these failed events to either the request queue or the response queue using the failed event manager.

About this task

To resubmit a failed SCA event, perform the following steps.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the Search Results page, select the check box next to each failed event you want to resubmit.
3. Click **Resubmit** or **Resubmit with trace**.
4. If the Process Response event qualifier is defined for the failed event, a resubmit page will appear. Select **Resubmit requests to the destination** or **Resubmit the exception response to the source**. Selecting **Resubmit the exception response to the source** allows the event to be sent to the response queue without having to be reprocessed.

Results

Depending on whether you selected **Resubmit requests to the destination** or **Resubmit the exception response to the source**, the event will be resubmitted to the appropriate queue.

Managing failed SCA events

When problems processing a Service Component Architecture (SCA) request or response message create a failed SCA event in the Recovery subsystem, you must decide how to manage that event. Use the information in this topic to help you identify and fix the error and clear the event from the Recovery subsystem.

About this task

Failed SCA events typically have source and destination information associated with them. The source and destination are based on the failure point (the location where the invocation fails), regardless of the type of interaction. Because runtime exceptions are not declared as part of the interface, component developers should attempt to resolve the exception and thus prevent a runtime exception from inadvertently being propagated to the client if the client is a user interface.

To manage a failed SCA event, perform the following steps.

Procedure

1. Use the failed event manager to locate information about the failed SCA event, taking note of the exception type.
2. Locate the exception type in Table 53 on page 575 to determine the location and possible causes of the error, as well as suggested actions for managing the failed event.

Table 113. Failed SCA events

Exception type	Possible cause of error	Suggested action
ServiceBusinessException	A business exception occurred during the execution of a business operation.	Look at the exception text to determine the exact cause, and then take appropriate action.
ServiceExpirationRuntimeException	A SCA asynchronous message has expired.	Set the expiration time using the RequestExpiration qualifier on the service reference. Investigate why the service is not responding fast enough.
ServiceRuntimeException	A runtime exception occurred during the invocation or execution of a service.	Look at the exception text to determine the exact cause, and then take appropriate action.
ServiceTimeoutRuntimeException	Response to an asynchronous request was not received within the configured period of time.	Set the expiration time using the RequestExpiration qualifier on the service reference. Investigate why the service is not responding fast enough.
ServiceUnavailableException	This exception is used to indicate that there was an exception thrown while invoking an external service via an import.	Look at the exception text to determine the exact cause, and then take appropriate action.

Table 113. Failed SCA events (continued)

Exception type	Possible cause of error	Suggested action
ServiceUnwiredReferenceRuntimeException	A SCA reference used to invoke a service is not wired correctly.	Look at the exception text to determine the exact cause, and then take appropriate action to correctly wire the SCA reference.

Managing failed JMS events

When problems processing a JMS request or response message create a failed JMS event in the Recovery subsystem, you must decide how to manage that event. Use the information in this topic to help you identify and fix the error and clear the event from the Recovery subsystem.

About this task

To manage a failed JMS event, perform the following steps.

Procedure

1. Use the failed event manager to locate information about the failed JMS event, taking note of the exception type.
2. Locate the exception type in Table 54 on page 576 to determine the location and possible causes of the error, as well as suggested actions for managing the failed event.

Table 114. Failed JMS events

Exception type	Location of error	Possible cause of error	Suggested action
FaultServiceException	Fault handler or fault selector	There is malformed data in the JMS message.	1. Inspect the JMS message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the fault handler or fault selector.	1. Debug the custom fault selector or fault handler, fixing any errors identified. 2. Resubmit the failed event.
ServiceRuntimeException	Fault handler	The fault selector and runtime exception handler are configured to interpret the JMS message as a runtime exception. This is an expected exception.	Look at the exception text to determine the exact cause, and then take appropriate action.

Table 114. Failed JMS events (continued)

Exception type	Location of error	Possible cause of error	Suggested action
DataBindingException or DataHandlerException	Data binding or data handler	There is malformed data in the JMS message.	1. Inspect the JMS message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the data binding or data handler.	1. Debug the custom data binding or data handler, fixing any errors identified. 2. Resend the message. 3. Delete the failed event.
SelectorException	Function selector	There is malformed data in the JMS message.	1. Inspect the JMS message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the function selector.	1. Debug the custom function selector, fixing any errors identified. 2. Resend the message. 3. Delete the failed event.

Managing failed WebSphere MQ events

When problems processing a WebSphere MQ request or response message create a failed WebSphere MQ event in the Recovery subsystem, you must decide how to manage that event. Use the information in this topic to help you identify and fix the error and clear the event from the Recovery subsystem.

About this task

To manage a failed WebSphere MQ event, perform the following steps.

Procedure

1. Use the failed event manager to locate information about the failed event, taking note of the exception type.
2. Locate the exception type in Table 55 on page 578 to determine the location and possible causes of the error, as well as suggested actions for managing the failed event.

Table 115. Failed WebSphere MQ events

Exception type	Location of error	Possible cause of error	Suggested action
FaultServiceException	Fault handler or fault selector	There is malformed data in the WebSphere MQ message.	1. Inspect the message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the fault handler or fault selector.	1. Debug the custom fault selector or fault handler, fixing any errors identified. 2. Resubmit the failed event.
ServiceRuntimeException	Fault handler	The fault selector and runtime exception handler are configured to interpret the WebSphere MQ message as a runtime exception. This is an expected exception.	Look at the exception text to determine the exact cause, and then take appropriate action.
DataBindingException or DataHandlerException	Data binding or data handler	There is malformed data in the WebSphere MQ message.	1. Inspect the message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the data binding or data handler.	1. Debug the custom data binding or data handler, fixing any errors identified. 2. Resend the message. 3. Delete the failed event.

Table 115. Failed WebSphere MQ events (continued)

Exception type	Location of error	Possible cause of error	Suggested action
SelectorException	Function selector	There is malformed data in the WebSphere MQ message.	1. Inspect the message and locate the malformed data. 2. Repair the client that originated the message so it creates correctly formed data. 3. Resend the message. 4. Delete the failed event.
		There was an unexpected error in the function selector.	1. Debug the custom function selector, fixing any errors identified. 2. Resend the message. 3. Delete the failed event.

Finding Common Base Events related to a failed event

A failed event can be related to one or more Common Base Events. The failed event manager provides a link to view related Common Base Events in the Common Base Event Browser.

Before you begin

You must be logged in as administrator or operator to perform this task.

About this task

Examining related Common Base Events can give you additional information about how or why the original event failed. The failed event and any related Common Base Events are linked by the same session ID.

To find and view related Common Base Events, use the following procedure.

Procedure

1. From within the administrative console, use the failed event manager to locate the failed event you want to investigate. See "Finding failed events in WebSphere ESB" on page 566 for instructions on how to search for failed events.
2. From the Failed Event Details page for that event, click **Browse Related Common Base Events**.

Results

The Common Base Event Browser opens in a new browser window and lists any Common Base Events related to the original failed event.

Deleting failed events

If you do not want to resubmit a failed event, or if you have failed events that have expired, use the failed event manager to delete them from the server. The failed event manager provides three options for deleting failed events.

Before you begin

You must be logged in as administrator or operator to perform this task.

About this task

To delete one or more failed events, use the following procedure.

Procedure

1. Ensure that the failed event manager is open and that you have retrieved a list of the failed events on your system.
2. From the failed event manager's Search Results page, do one of the following:
   - If you want to delete one or more specific failed events, select the check box next to each event and then click **Delete**.
   - If you want to delete only those failed events that have expired, click **Delete expired events**. Note that this deletes only the expired events in the current set of search results.
   - If you want to delete all failed events on the server, click **Clear all**.

Troubleshooting the failed event manager

This topic discusses problems that you can encounter while using the failed event manager.

Select the problem you are experiencing from the table below:

Problem	Refer to the following
I am having trouble entering values in the Search page's **By Date** tab	"Values in the By Date and From Date field automatically change to default if entered incorrectly" on page 580
I am having trouble deleting expired events	"Using the Delete Expired Events function appears to suspend the failed event manager" on page 581
I am having trouble with failed events not being created	"Failed events are not being created" on page 581
I am having trouble retrieving or deleting large numbers of failed events	"The server fails when retrieving or deleting large numbers of failed events" on page 581

Values in the By Date and From Date field automatically change to default if entered incorrectly

The Search page's **From Date** and **To Date** fields require correctly formatted locale-dependent values. Any inconsistency in the value's format (for example, including four digits in the year instead of 2, or omitting the time) will cause the failed event manager to issue the following warning and substitute a default value in the field:

```
CWMAN0017E: The date entered could not be parsed correctly:
your_incorrectly_formatted_date. Date: default_date is being used.
```

The default value of the **From Date** field is defined as `January 1, 1970, 00:00:00 GMT`.

Important: The actual default value shown in your failed event manager implementation will vary depending on your locale and time zone. For example, the From Date field defaults to `12/31/69 7:00 PM` for a workstation with an en_US locale in the Eastern Standard Time (EST) time zone.
The default value for the **To Date** field is always the current date and time, formatted for your locale and time zone.

To avoid this problem, always enter your dates and times carefully, following the example provided above each field.

Using the Delete Expired Events function appears to suspend the failed event manager

If you use the Delete Expired Events button in situations where there are many failed events in the current search results, or where those events contain a large amount of business data, the failed event manager can appear to be suspended indefinitely.

In this situation, the failed event manager is not suspended: it is working through the large data set, and will refresh the results set as soon as the command completes.

Failed events are not being created

If the Recovery subsystem is not creating failed events, go through the following checklist of potential causes:

- Ensure that the wpsFEMgr application is running. If necessary, restart it.
- Ensure that the failed event manager's database has been created, and that the connection has been tested.
- Ensure that the necessary failed event destination has been created on the SCA system bus. There should be one failed event destination for each deployment target.
- Ensure that the Quality of Service (QoS) **Reliability** qualifier has been set to `Assured` for any Service Component Architecture (SCA) implementation, interface, or partner reference that participates in events you want the Recovery service to handle.

The server fails when retrieving or deleting large numbers of failed events

The server can fail if you use the failed event manager to retrieve or delete a large number of failed events at once. To prevent this problem, be sure to check the total failed event count on the main page of the failed event manager before performing search or delete actions. If you have a large number of events, do not try to retrieve or delete all of them at once. Instead, use the **Search failed events** option to return a subset of failed events that match specific criteria. You can then delete all of the events in the filtered result set without causing a server failure.

Note: There are several factors in your environment that can affect the number of failed events the server can return or delete in a single request, including other processes running on the same machine as the server and the amount of available memory.

Chapter 25. Recovering from a failure

Recovering from a failure requires an understanding of standard system processing in the event of a failure, as well as an understanding of how to analyze problems that may be the cause of a failure.

Overview of the recovery process

The recovery process encompasses a set of tasks that include both analysis and procedures.

When you must recover from a failure, these are the high-level steps to follow:

- Familiarize yourself with the possible kinds of failures. See *Triggers for recovery* for more information.
- Assess the state of the system. See *Assessing the state of the system* for more information.
- Form a hypothesis about what the problem is.
- Collect and analyze the data.
- Refer to other topics in this information center for instructions on fixing the problem.

Triggers for recovery

The need for solution recovery can result from a variety of triggers.

Situations from which solution recovery is necessary

Solution recovery is the process of returning the system to a state from which operation can be resumed. It encompasses a set of activities that address system failure or system instability that can be triggered by unforeseen circumstances.

You may need to perform solution recovery activities for the following circumstances:

- **Hardware failure**

  Abnormal termination or system down can be caused by a power outage or catastrophic hardware failure. This can cause the system (all if not most JVMs) to stop.

  In the case of a catastrophic hardware failure, the deployed solution may enter an inconsistent state on restart.

  Hardware failures and environmental problems also account for unplanned downtime, although by far not as much as the other factors.

  You can reduce the likelihood of hardware failures and environmental problems by using functions such as state-of-the-art LPAR capabilities with self-optimizing resource adjustments, Capacity on Demand (to avoid overloading of systems), and redundant hardware in the systems (to avoid single points of failure).
- **System is not responsive**

  New requests continue to flow into the system but on the surface it appears that all processing has stopped.
- **System is unable to initiate a new BPEL process instance**

© Copyright IBM Corp. 2007, 2011

The system is responsive and the database seems to work correctly. Unfortunately, the new BPEL process instance creation is failing.

- **Database, Network or Infrastructure Failure**

  In the case of fundamental infrastructure failure, the solution may require administration to restart/resubmit business transactions after the infrastructure failure is resolved.

- **Poor tuning or a lack of capacity planning**

  System is functional but is severely overloaded. Transaction time-outs are reported and there is evidence of an overflow of the planned capacity.

  Incomplete capacity planning or performance tuning can cause this type of solution instability.

- **Defects in application module development**

  The modules that are part of a custom developed solution can have bugs. These bugs can result in solution instability and failed services.

  Bugs in a custom developed solution can result from a variety of situations, including (but not limited to) the following:

  – Business data that was not planned for or unforeseen by the application design.

  – An incomplete error handling strategy for the application design.

    A detailed error handling design can reduce solution instability.

- **WebSphere software defect**

  A defect in the WebSphere product causes a backlog of events to process or clear.

Assessing the state of the system

The first thing to do when an abnormal condition occurs is to take the *pulse* of the overall system and get a feel for how much or how little of the system is operational and how much of it is rendered 'out of service' by whatever the external stimuli was that caused this condition.

Address a predefined set of questions to assess the extent of the outage. The following list provides examples of predefined questions designed to help you gather the appropriate information:

1. Is this system still performing work?

   Determine if system is still operational. Often times, a system can be operational, but because of overload or inappropriate tuning, or both, the system is not completing tasks quickly and/or is attempting to do work that is indeed failing.

   The litmus test for each of these questions will be specific to the nature of the deployed solution.

2. What special error handling support is built-in to the application?

   If there is a lot of automated retry and various support logic, then the application itself might shield some errors from manifesting the IT operator.

   These conditions must be known and documented for reference by the recovery team.

Things you can do to help assess the state of the system include the following:

1. Check to see if the server is at least running.

   Do you see the PID or get a positive feedback from the deployment manager via the administrative console?

2. Check to see if there are locks in the database(s) or any unusual database traffic.

   Most databases will have facilities to look at locks. Depending on the deployment topology, there maybe multiple databases.

   - Messaging Engine Database
   - Business Process Container Database
   - WebSphere Process Server Common Database (Failed Events and Relationship data)
3. Check to see what the status of the messaging system is.

   Check for events or messages in the following locations:

   - Business Process Choreographer Hold and Retention Destinations
   - Number of failed events
   - Number of messages on the solutions module destinations
4. Check to see if the database is functioning.

   Can you perform some simple SELECT operation, on unlocked data in a reasonable amount of time?
5. Check to see if there are errors in the database log.

If the database is not working properly, then recovering the database (so that it can at least release locks and perform simple selects) is vital to system recovery.

If the messaging system is not working properly, then recovering the messaging subsystem, so that it can at least be viewed and managed, is also vital to system recovery.

Note: A 'bottoms up' approach is not always conclusive. However, chances of successful recovery vary based on these basic activities.

From these basic procedures and health check kinds of activities, start to look at some specific situations. Patterns will be described, specifics will be given and insights as to what is going on under the covers will be provided.

Realize that this situational analysis is a read-only activity. While it provides vital information from which to determine the appropriate recovery actions, it should not change the state of the system under review. It is impossible to predict and provide prescriptive actions for all possible causes of a system outage. For example, consider the following decision tree:

There are many broad categories to investigate in the event of an unplanned outage. These broad categories will have sub categories and so on. Defining prescriptive actions for each node and the subsequent node will depend on the results of each investigation. Because this type of relationship is difficult to convey in a document form, using a support tool such as *IBM Guided Activity Assist* to interactively walk you through the investigative and decision making process is recommended. As you progress from the top to each child node, it is important to conduct the appropriate level of situational analysis.

Recovery: Analyzing the problem

For all unplanned system events, a set of basic recovery procedures can be leveraged at the point of identification.

There are six well defined steps to situational analysis:

1. Define the question
2. Gather information and resources (observe)
3. Form hypothesis
4. Perform experiment and collect data
5. Analyze data
6. Interpret data and draw conclusions that serve as a starting point for new hypothesis

For each production scenario the symptoms that initiate a recovery action may vary.

It is important to follow the guidelines for situational analysis and take the corrective action relative to the symptoms that are presented.

Situational analysis

Situational analysis is the cyclical execution of the scientific method and can take into account various situations that will initiate a recovery procedure.

The following list is of the different types of situations that will initiate a recovery procedure:

- Abnormal termination or system down

  A power outage or catastrophic hardware failure has caused the system (all if not most JVMs) to stop.
- System is not responsive

  New requests continue to flow into the system but on the surface it appears that all processing has stopped.
- System is functional but is severely overloaded

  Transaction time-outs are reported and there is evidence of an overflow of the planned capacity.
- System is unable to initiate new process instance

  The system is responsive and the database seems to work correctly. Unfortunately, new process instance creation is failing.

Recovery: First steps

Administrators can facilitate solution recovery processes by following a first steps checklist of general practices.

The following list describes actions that you **SHOULD NOT TAKE** under normal circumstances when trying to recover a solution.

Note: There could be special situations for which you might need to perform some of the actions listed in this topic. However, you should never initiate any of these actions without first consulting with the WebSphere ESB support organization.

- Do not delete the transaction log file

  The transaction (tranlog) log file stores critical transactional data that is written to databases. It is an internal file that WebSphere Application Server uses to manage in-flight transactions and attempt to recover them should the server lock up.
- Do not have the transaction logs local on the cluster members

  Put the transaction logs on a shared drive. This is the only way to allow peer recovery, which helps minimize the downtime during recovery.
- Do not attempt database operations where the result set is large enough to create additional resource contention (OutOfMemory)
- Avoid performing Business Process Choreographer Explorer operations that return large result sets.
- Avoid executing administrative scripts on process instances without considering the result set size.
- Do not drop and/or re-create databases in production
- Do not uninstall applications as part of your standard recovery procedures

  You should only uninstall applications with the direction from the IBM support organization.

- Do not enable too much trace if the system is overloaded.

  Too much trace will cause a slowdown in system throughput and might cause transaction time-outs. Too much trace can often add to the problems that need to be addressed, rather than providing insight as to how to solve the original problems.

  Get immediate assistance from IBM support to define the correct trace specification.
- Do not experiment or try out new scripts or new commands on production systems.
- Do not run your production servers in *development mode*

  Enabling the **Run in development mode** option may reduce the startup time of an application server. This may include JVM settings such as disabling bytecode verification and reducing JIT compilation costs.

The following list describes the recommended actions when it comes to recovery.

- Always take a *snapshot* of the configuration tree, the PI file of the application in question and the logs which are available.

  Logs may be overwriting themselves depending on the configuration. Capturing a set early and often is an important step for postmortem analysis. See the topic on *IBM Support Assistant (ISA)* for details on the IBM Support Assistant, which helps with this type of activity.
- Always understand your database settings, especially related to database transaction log file size, connection pools, and lock timeouts.

Failed-event locations: Where does the data go?

For all (production and test) recovery activities there are a finite number of locations in the solution where events accumulate.

By adhering to guidelines and preventive measures described in *Planning error prevention and recovery*, all business events and associated data will reliably accumulate in one of these locations.

If you do not adhere to sound architectural and application development practices, then a percentage of inflight events may end up in an inconsistent state, from which recovery cannot be attained. Under such circumstances, (presumably

identified during testing cycles) post recovery investigation and clean up is necessary to correct the issue so that future recovery activities are completely successful.

In order to accurately describe the following scenarios, it is important to put the information in the context of a use case.

Use case: recovering data from failed events

A use case provides a context for a recovery scenario. In the use case, a business has an application that receives a request to create a new Account.

The solution is comprised of multiple modules as recommended through module best practices.

The first module mediates the request and delegates work to an Account Creation process. In the Figure 319 we have implemented the solution as separate modules where the request is passed between the mediation module (AccountRouting) and the processing module (AccountCreation) via an SCA import/export. See the following screen capture for an illustration of the two modules.

Figure 319. Assembly diagram of account routing process

From the assembly diagram shown in Figure 319, you can begin to see at what locations in the flow that failures might occur. Any of the invocation points in the

assembly diagram can propagate or involve a transaction. There are a few areas in the flow where data will collect as a result of application or system failures.

In general, transaction boundaries are created and managed by the interaction (synchronous and asynchronous) between components and import/export bindings and their associated qualifiers. Business data accumulates in specific recovery locations most often due to transaction failure, deadlock or rollback.

Transaction capabilities within WebSphere Application Server help WebSphere ESB enlist transactions with service providers. These enlisted interactions are particularly important to understand with respect to import and export bindings. Understanding how imports and exports are used within your specific business cases is important in determining where events in need of recovery accumulate.

An error handling strategy should define interaction patterns, transactions used, import and export usage before developing the application. The solution architect should identify the preferences to use, guidelines, that are then used as the application is created. For example, the architect needs to understand when to use synchronous vs. asynchronous calls, when to use BPEL fault handling and so forth. The architect needs to know whether or not all services can participate in transactions, and for those services that can not participate, how to handle compensation if problems are encountered.

Additionally, the application shown in the assembly diagram in Figure 319 on page 1101 leverages connectivity groups and module development best practices. By leveraging this pattern we now have the ability to stop the inbound flow of new events by stopping the `AccountRouting` module.

The following sections address the location of business data in the case of failure and recovery.

Business Flow Manager or Human Task Manager

In our business case, we leverage a BPEL process for `AccountCreation` process.

With regard to recovery, there are a some questions you need to ask yourself with respect to BPEL and human task management:

1. What type of process is being run (short running or long running, business state machine, human task) ?

   Short running processes are known as microflows.
2. Is the process developed properly and using fault handling to promote data integrity?
3. How are the invocation patterns and unit of work properties configured to predict and control transaction boundaries?

Knowing the answers to these questions will impact your recovery strategy for invocations 7 and 8 shown in the assembly diagram, as highlighted in Figure 320 on page 1103.

Figure 320. Assembly diagram of account routing - invocations 7 and 8

Stateful components, such as long-running BPEL processes and business state machines, involve many database transactions where process activity changes and state changes are committed to the database. The work progresses by updating the database and placing a message on an internal queue that describes what is to be done next. More information on macro flow transactions is available in the information center topic titled *Transactional behavior of long-running processes*.

If there are problems processing messages that are internal to the Business Flow Manager, these messages are moved to a *Retention Queue*. The system attempts to continue to process messages. If a subsequent message is successfully processed, the messages on the retention queue are resubmitted for processing. If the same message is placed on the retention queue five times, it is then placed on the hold queue. Information, such as what internal queues are used or retry algorithms for these queues is described in detail in the information center topic titled *Recovery from infrastructure failures*.

Additional information about viewing the number of messages and replaying messages can be found in Replaying Messages from the Retention Queue / Hold Queue.

Failed event manager

The Failed event manager (FEM) is used to replay events or service invocation requests that are made asynchronously between *most* component types.

Failed events are created if the `AccountRouting` component makes an asynchronous call to the SCA Import binding `AccountCreationSCAImport` and a `ServiceRuntimeException` is returned.

It is important to note that failed events are not generated in most cases where BPEL is the client in the service interaction. This means that the invocation for 7 and 8 (as shown in Figure 320 on page 1103) will not typically result in a failed event. BPEL provides fault handlers and other ways to model for failure. For this reason, if there is a `ServiceRuntimeException` (SRE) failure calling "JDBCOutboundInterface", the SRE is returned to the BPEL for processing. The error handling strategy for the project should define how runtime exceptions are consistently handled in BPEL.

It is important to note however, failed events are created for asynchronous response message for the BPEL client if these messages can not be delivered to the process instance due to an infrastructure failure.

The following diagram illustrates how the Failed event manager component works. Descriptions of the processing associated with each numbered step are provided in Figure 321.

Figure 321. Failed event manager processing

Failed event manager processing

1. The source component makes a call using an asynchronous invocation pattern
2. The SCA MDB picks the message up off the SCA destination
3. The SCA MDB makes the call to the correct target component
4. The target component throws a `ServiceRuntimeException`
5. The SCA MDB transaction rolls back to the SCA destination
6. The exception information is stored to the Failed event manager database with a status of *not confirmed*
7. The invocation is retried by the SIBus n number of times

The retry limit default value is 5 - one original and 4 retries. You can change the default value in the administrative console. For example, given an SCA module M, you could navigate to **Buses** > **SCA.SYSTEM.<CELL>.BUS** > **Destinations** > **sca/M** and change the value in the *Maximum failed deliveries* field.

8. After the number of retries reaches the specified limit, the message is moved to the FEM destination.
9. The Failed Event Manager database picks up the message
10. The Failed Event Manager database updates the failed event in the database and the status is set to *failed*.

When are "failed events" created?

As stated, failed events are neither created for synchronous invocations nor typically for two-way business process interactions.

Failed events are generally created when clients use an asynchronous invocation pattern and a ServiceRuntimeException is thrown by the service provider.

If everything is done synchronously and in the same transaction, data is not collected anywhere. Instead it is all rolled back to the client that made the call. Where ever a commit is occurs, data collects. If the calls are all synchronous, but there are multiple commits, then these commits become an issue.

In general, you should use asynchronous processing calls or long running BPEL if multiple transactions are needed. So each ASYNC call is a chance for data to collect. Long running BPEL process are a collection point.

Table 116. Invocation patterns and relationship to the creation of failed events: **Service Business Exceptions**

Invocation Pattern	Failed Event Created Y/N?	Notes
Synchronous	No	Failed events are not created for service business exceptions or when using a synchronous pattern
Asynchronous - One Way	No	By definition, one-way invocations cannot declare faults, meaning, it is impossible to throw a ServiceBusinessException.
Asynchronous - Deferred Response	No	Failed Events are not created for service business exceptions
Asynchronous - Callback	No	Failed Events are not created for service business exceptions

Table 117. Invocation patterns and relationship to the creation of failed events: **Service Runtime Exceptions**

Invocation Pattern	Failed Event Created Y/N?	Notes
Synchronous	No	Failed events are not created for service runtime exceptions or when using a synchronous pattern.
Asynchronous - One Way	Yes	
Asynchronous - Deferred Response	Yes	
Asynchronous - Callback	Yes	
BPEL - Two Way	No	Failed Events are not created when the source component is a business process. **Note:** For an asynchronous call, if the response can not be returned to BPEL, then a failed event is created.

Table 117. Invocation patterns and relationship to the creation of failed events: **Service Runtime Exceptions** *(continued)*

Invocation Pattern	Failed Event Created Y/N?	Notes
BPEL - One Way	Yes	

For additional information, review the information center topic titled *Managing failed events*.

Additional information about viewing and resubmitting failed events can be found in section *Resubmitting Failed Events*.

Service Integration Bus Destinations

Messages that are waiting to be processed may accumulate in a few Service Integration Bus (SIBus) destinations. For the most part these destinations are "system" destinations. Messages within these destinations typically are a mixture of three types:

- Asynchronous requests for processing
- Asynchronous replies to requests
- Asynchronous messages that failed deserialization or function selector resolution

Note: Asynchronous replies can be valid Business Objects or faults returned as a result of a request.

SCA Module Destination

Again, referring back to our business case.

There would be two "SCA Module" destinations in the solution:

- sca/AccountRouting
- sca/AccountCreation

These destinations are created when the module is deployed to an application server or a cluster.

There are rare opportunities for messages to accumulate in these destinations. The accumulation of messages in these locations is a strong indication that there maybe a performance problem or an application defect. Investigate immediately. It is important to monitor the depth of the module destinations (with your chosen IT monitoring solution) as a back up of messages could lead to a system outage or a prolonged recycle time.

We call these "SCA Module" destinations because the generated name is the same as the module name with the additional "sca/". These destinations are pivotal in the functioning of SCA asynchronous invocations (brokering requests and responses). There are a varying number of additional destinations that are generated during application installation on the SCA.SYSTEM bus but for the purpose of the discussion we'll be addressing the importance of the "SCA Module" destination.

System Integration Bus Retry

As we learned above, the FEM has a built-in retry mechanism with the SCA message driven bean (MDB). This retry behavior can be controlled by modifying the “Maximum Failed Deliveries” attribute on the module destination.

Note: Typically, there is no reason to adjust this retry capability. The information provided here is for completeness.

Referring to our business case, there are a number of SI Bus destinations created by SCA to support asynchronous communication.

As we have learned, one of these destinations is called “sca/AccountRouting ” You can adjust the number of retries that happen during a ServiceRuntimeException of an asynchronous service invocation by changing the value of the “Maximum Failed Deliveries” property via the admin console. However, you may not set the value less than 2 in modules with a BPEL process. The second delivery is required to return ServiceRuntimeExceptions back to the BPEL for processing.

System Exception Destinations

The failed event manager is one place where we can look to administer failures. When dealing with Imports and Exports that are JMS or EIS based, we have to consider another important location.

Destinations on the SCA.Application bus are configured to route failed messages to the SIB system exception destination for that bus. Thus, if a JMS export picks up a message from the SCA Application bus and runs into a rollback situation, the failed message will be routed to the SIB system exception destination instead of the WBI recovery exception destination. This scenario differs from the failed event discussion above in that a failure to deserialize a message on the SCA.Application bus will not result in a failed event. There is a system exception destination on every bus within the solution. These destinations must be monitored and administered much like the “dead letter queue” common to MQ infrastructures.

Consider the following scenario.

An external JMS client places a message on an inbound queue exposed via a JMS Export The JMS Export Binding MDB picks up the message for processing. From here, one of two things will happen:

1. The JMS export successfully parses the message and determines which operation on the interface to invoke at which point the message is sent to the SCA runtime for processing.
2. The JMS export fails to recognize the message body as a valid business object or the JMS export binding *deserializes* the message body but is unable to determine the appropriate operation on the interface to invoke. At this point the message is placed on the System Exception Destination for the bus.

We can have this type of failure when trying to receive requests from the `AccountRoutingJMSExport` (1). This export is a JMS export and there is a possibility that events can accumulate on the System Exception Destination on the SCA.Application.Bus. Use the chosen IT monitoring solution to observe the depth of this destination.

Failed Event Manager and SIB Destinations

For WebSphere ESB, the exception destination is set to the WebSphere ESB exception destination queue. This queue follows a naming convention as follows:

```
Node name: WESBNode
Server name: server1
Recovery exception destination: WBI.FailedEvent.WESBNode.server1
```

In general, all the destinations created on the SCA.System bus will be configured to route failed messages to the recovery exception destination.

When a system failure occurs, in addition to capturing the failed message in this exception destination, the WebSphere ESB recovery feature also generates a failed event that represents the system error and stores it into the Recovery database as described in the Failed Event Manager section of this document.

Summary

In summary, WebSphere ESB provides administrative capabilities above and beyond the underlying WebSphere Application Server platform. Proper measures should be made to understand and use these capabilities along with following the guidance provided in the Planning error prevention section of *Planning error prevention and recovery*.

Table 118. Administrative capabilities to help manage failures

Administrative Capability	Bundled With WebSphere ESB Y/N?	Summary
Failed Event Manager	Yes	Read/Edit/Delete Access. This is the central place to administer Service Runtime Exceptions and other forms of infrastructure failures.
Service Integration Bus Browser	Yes	Read/Delete. Use the Service Integration Bus Browser on the administrative console for browsing and performing day-to-day operational tasks on service integration buses.

Note: The number of events or records that can be simultaneously administered by these tools are specific to external factors such as memory allocation, result sets

and DB tuning, connection timeout. Run tests and set the appropriate thresholds to avoid exceptions (OOM, TransactionTimeOut).

Recovery troubleshooting tips

This section provides a list of tips for troubleshooting the recovery process.

Restarting deployment environments

As one step in a recovery process, you may need to restart of you deployment environment.

About restarting deployment environments

The procedure to restart a deployment environment varies depending on the topology. Topologies are based on system configuration patterns, each pattern designed to meet particular business requirements.

WebSphere ESB supports a set of predetermined deployment environment configuration patterns. If none of the patterns meet your requirements, you can plan and create your own customized deployment environment.

In any given deployment environment configuration pattern there are a number of servers running as JVM processes. In general there are three types of servers as follows:

- Messaging Servers

  Messaging servers are responsible for providing the Service Integration Bus (SIB) messaging infrastructure.
- WebSphere ESB Servers

  Servers with profiles capable of only hosting and running mediation modules.
- WebSphere Process Servers

  Servers with profiles capable of hosting and running all module types. This profile hosts the Business Process Choreographer component.
- Support Servers

  This server is responsible for providing support and monitoring services such as the Common Event Infrastructure CEI.

The deployment patterns differ in how you group and organize all the functional components, so that the pattern can address your business requirements in the most cost effective fashion. For more advanced and highly available environments, the servers would reside in clusters that are distributed across physical resources.

General practice for restarting servers as part of a recovery operation

A general model for starting servers is to start the messaging servers first, then the support servers and lastly the WebSphere ESB servers. Each application architecture may have specific dependencies between application components that need to be taken into consideration.

Shutdown basically happens inverse to the startup procedure, starting with the application server clusters and ending with shutting down the messaging infrastructure after it has had time to quiesce and process any inflight transactions.

Related tasks:

Choosing your deployment environment pattern
You can configure your deployment environment by choosing one of the IBM-supplied topology patterns or by creating your own custom deployment environment. This topic section lists and describes the available IBM-supplied topology patterns and presents considerations for choosing a topology.

Related information:

IBM Business Process Manager and WebSphere Enterprise Service Bus deployment patterns: Selecting your deployment pattern

Viewing the service integration bus

Use the Service Integration Bus browser on the administrative console to view the service integration bus.

Before you begin

Make sure you understand how the Service Component Architecture (SCA) system bus is used.

About this task

The Service Integration Bus Browser provides a single location for browsing and performing day-to-day operational tasks on service integration buses.

Viewing the service integration bus is a useful way to determine if messages are accumulating on the SCA Module destinations.

The accumulation of messages on the SCA Module destinations is a strong indication that there maybe a performance problem or an application defect.

It is a good idea to periodically view the messages and determine if there are any messages have become locked for an extended duration of time as this may indicate that there are "indoubt transactions".

Procedure

1. From the administrative console, expand **Service integration**.
2. Select **Buses**.

3. Select the appropriate messaging bus for the service. The following example shows the messaging bus named `SCA.System.cleanup1cell01.bus` highlighted, where `cleanup1cell01` is the name of the cell.

4. Select **Destinations**

5. Review the relevant information. You should look at the destinations named `sca/XYZ`, where `XYZ` is the name of the module. For example, for modules named `AccountRouting` and `AccountCreation`, you would look for the following destinations:

6. Select the link text for the destination that you are interested in viewing.
   This will link you to a general properties page for the destination that you want to view.
7. From the general properties page of the destination, select the **Queue points**

8. From the Queue points page, select the link for the message point.

9. Select the **Runtime** tab.
   From this screen you can see the current message "depth" and the threshold. Selecting the **Messages** link will allow you to view the message contents.

Ideally, use an appropriate IT monitoring tool and set alert thresholds for these destinations. The threshold value would be established during the performance test phase for the application.

Messages on a production system should never be deleted unless explicitly directed to do so by the SCA L3 team.

Related concepts:

"Service Integration Bus Browser" on page 479
The Service Integration Bus Browser provides a single location for browsing and performing day-to-day operational tasks on service integration buses.

Related tasks:

"Resolving indoubt transactions" on page 1115
Transactions can become stuck in the indoubt state indefinitely due to exceptional circumstances, such as the removal of a node causing messaging engines to be destroyed.

Related information:

Considerations for Service Component Architecture support in servers and clusters

Capturing javacore

There are a number of methods that you can use to capture a javacore from an IBM JDK and thread dumps for non-IBM JDKs.

Capturing javacore

A javacore dump, or a thread dump as it is also called, is one of the primary problem determination documents that an application server creates.

1. Use wsadmin to produce a javacore in the Profile directory:
   a. For Windows:

```
<PROFILE_DIR>\bin\wsadmin.bat [-host host_name] [-port port_number]
[-user userid -password password] -c
"$AdminControl invoke [$AdminControl queryNames WebSphere:name=JVM,process=server1,*]
dumpThreads"
```

b. For Unix (IBM JDKs):

```
<PROFILE_DIR>>/bin/wsadmin.sh[-host host_name]
[-port port_number] [-user userid -password password] -c
"\$AdminControl invoke [\$AdminControl queryNames WebSphere:name=JVM,process=server1,*]
dumpThreads"
```

Note: The braces, [] around the `AdminControl queryNames` command are part of the command, and not used to signify optional parameters as is the case for braces around host, port and user. The process name: server1 may need to be change to fit your configuration.

2. A signal can be sent to the server process:
   a. Windows:

      A launch script must be used to start the server process to allow the signal to be passed to the process. This does require special setup before starting the server.

      1) <PROFILE_DIR>\bin\startServer.bat server1 –script SERVER1.bat
      2) b. SERVER1.bat

         The server process will start in a command window. You will need to check the logs to verify that the server has successfully started since the intermediate JVM process which usually starts the server process is not used.
      3) <CTRL><BREAK>

         Issue a <CTRL><BREAK> into the command window where the server process is running. A javacore will be produced.

   b. UNIX (all JDKs): kill -3 <pid>

      Where <pid> is the process id of the WebSphere Process Server. For IBM JDKs a javacore will be produced in the <PROFILE_DIR>directory.

      For non-IBM JDKs, a thread dump will be written to the native_stdout.log.
3. An alternative method to dumping a windows core file is to use jvmdump.

   This does not require special setup before starting the server. However, it does require a special executable from the JVM team. The jvmdump.exe program can be requested by sending a note to jvmcookbook@uk.ibm.com. The advantage of this method is additional information can be obtained about native code being executed within JVM. The format of the dump differs from the IBM javacores.

   - jvmdump.exe <PID>
   - `<WAS_HOME>>\java\jre\bin\jextract.exe <core.name.dmp>`
   - `<WAS_HOME>\java\jre\bin\jdumpview.exe`
     - set dump <core.name.dmp>.zip
     - display thread

       Displays the current executing thread at the time of the dump
     - c. display thread *

       Display all of the threads from the dump.

   For more details about the jdumpview utility consult the Diagnostics Guide for the IBM Developer Kit and Runtime Environment, Java Technology Edition, Version 5.0.

Related information:

Configuring the hang detection policy

Servers and recovery mode processing

When you restart an application server instance with active transactions after a failure, the transaction service uses recovery logs to complete the recovery process.

These recovery logs, which each transactional resource maintains, are used to rerun any Indoubt transactions and return the overall system to a self-consistent state. An *indoubt transaction* is one that has encountered environmental or other errors during commit processing. Logging occurs for normal inflight transactions, but those log entries are removed upon successful commit processing.

This recovery process begins as soon as all of the necessary subsystems within the application server are available during a server startup. If the application server is not restarted in recovery mode, the application server can start accepting new work as soon as the server is ready, which might occur before the recovery work has completed. This might be okay in many cases, but the more conservative option is provided here. To be clear, recovery will run on a server restart even if the server is started in 'normal' start model.

For information on how to start a server, see the Starting a server topic in the WebSphere ESB information center.

Related tasks:

"Starting managed servers" on page 382
Start the application server process in order to run applications on the managed server.

Resolving indoubt transactions

Transactions can become stuck in the indoubt state indefinitely due to exceptional circumstances, such as the removal of a node causing messaging engines to be destroyed.

Before you begin

Use the procedure to resolve indoubt transactions only if you have tried other procedures (such as restarting the server in recovery mode), unsuccessfully.

About this task

When a transaction is stuck in the indoubt state, it must either be committed or rolled back so that normal processing by the affected messaging engine can continue.

You can use the administrative console to display the messages causing the problem by Listing messages on a message point.

If there are messages related to an indoubt transaction, the identity of the transaction displays in a panel associated with the message. You can then resolve the transaction in one of the following ways:

- Using the server's transaction management panels
- Using methods on the messaging engine's MBean

You should first attempt to resolve the indoubt transaction using the application server transaction management panels. If this does not work, then use methods on the messaging engine's MBean. These are described in the Procedure section below.

Procedure

1. **Using the application server transaction management panels to resolve indoubt transactions**
   a. Navigate to the transaction management panels in the administrative console

      Click **Servers** > **Application servers** > **[Content Pane]** > *server-name* > **[Container Settings] Container Services** > **Transaction Service** > **Runtime** > **Imported prepared transactions - Review**
   b. If the transaction identity appears in the resulting panel, you can commit or roll back the transaction

      Choose the option to roll back the transaction

      If the transaction identity does not appear in the panel, the transaction identity was not enlisted with the Transaction Service on the server. In this case only, you should use methods on the MBean (as described in the next step) to display a list of the identities of the indoubt transactions managed directly by the messaging engine.
2. **Using methods on the messaging engine's MBean to resolve indoubt transactions**

   CAUTION:
   Only perform this step if you were unable to display the transaction identity by using the server's transaction management panels
   a. The following methods on the messaging engine's MBean can be used to get a list of transaction identities (xid) and to commit and roll back transactions:
      - **`getPreparedTransactions()`**
      - **`commitPreparedTransaction(String xid)`**
      - **`rollbackPreparedTransaction(String xid)`**
   b. To invoke the methods, you can use a wsadmin command, for example, you can use a command of the following form to obtain a list of the indoubt transaction identities from a messaging engine's MBean:

      ```
      wsadmin> $AdminControl invoke [$AdminControl queryNames type=SIBMessagingEngine,*] getPreparedTransactions
      ```

      Alternatively, you can use a script such as the following to invoke the methods on the MBean:

      ```
      foreach mbean [$AdminControl queryNames type=SIBMessagingEngine,*] {
        set input 0

        while {$input >=0} {
          set xidList [$AdminControl invoke $mbean getPreparedTransactions]

          set meCfgId [$AdminControl getConfigId $mbean]
          set endIdx [expr {[string first "(" $meCfgId]  - 1}]
          set me [string range ${meCfgId} 0 $endIdx]

          puts "----Prepared Transactions for ME $me ----"
          set index 0
          foreach xid $xidList {
            puts "  Index=$index XID=$xid"
            incr index
          }
          puts "------- End of list ---------"
      ```

```
    puts "Select index of XID to commit/rollback (-1 to continue) :"
    set input [gets stdin]

    if {$input < 0 } {
puts "No index selected, going to continue."
    } else {
      set xid [lindex $xidList $input]
      puts "Enter c to commit or r to rollback XID $xid"
      set input [gets stdin]
      if {$input == "c"} {
        puts "Committing xid=$xid"
        $AdminControl invoke $mbean commitPreparedTransaction $xid
      }
      if {$input == "r"} {
        puts "Rolling back xid=$xid"
        $AdminControl invoke $mbean rollbackPreparedTransaction $xid
      }
    }
    puts ""
  }
}
```

This script lists the transaction identities of the transactions together with an index. You can then select an index and commit or roll back the transaction corresponding to that index.

Results

In summary, to identify and resolve indoubt transactions:

1. Use the administrative console to find the transaction identity of indoubt transactions.
2. If a transaction identity appears in the transaction management panel, commit or roll back the transactions as required.
3. If a transaction identity does not appear in the transaction management panel, use the methods on the messaging engine's MBean. For example, use a script to display a list of transaction identities for indoubt transactions. For each transaction:
   a. Enter the index of the transaction identity of the transaction.
   b. Enter **c** to commit the transaction
   c. Enter **r** to roll back the transaction.
4. To check that transactions are no longer indoubt, restart the server and use the transaction management panel, or methods on the messaging engine's MBean.

Related tasks:

"Viewing the service integration bus" on page 1110
Use the Service Integration Bus browser on the administrative console to view the service integration bus.

Reviewing DB2 diagnostic information

Use a text editor to view the DB2 diagnostic log file on the machine where you suspect a problem to have occurred. The most recent events recorded are the furthest down the file.

About this task

Review DB2 diagnostic information when your systems are not working well. This is a way to see if the log files are full.

Procedure

On Unix type the following command: **tail -f /home/db2inst1/sqllib/db2dump/db2diag.log**
If the database is unresponsive, you will see something similar to the following:

```
2008-04-03-11.57.18.988249-300 I1247882009G504    LEVEL: Error
PID     : 16020                TID  : 3086133792  PROC : db2agent (WPRCSDB) 0
INSTANCE: db2inst1             NODE : 000         DB   : WPRCSDB
APPHDL  : 0-658                APPID: 9.5.99.208.24960.080403084643
AUTHID  : DB2INST1
FUNCTION: DB2 UDB, data protection services, sqlpWriteLR, probe:6680
RETCODE : ZRC=0x85100009=-2062548983=SQLP_NOSPACE
          "Log File has reached its saturation point"
          DIA8309C Log file was full.

2008-04-03-11.57.18.994572-300 E1247882514G540    LEVEL: Error
PID     : 16020                TID  : 3086133792  PROC : db2agent (WPRCSDB) 0
INSTANCE: db2inst1             NODE : 000         DB   : WPRCSDB
APPHDL  : 0-658                APPID: 9.5.99.208.24960.080403084643
AUTHID  : DB2INST1
FUNCTION: DB2 UDB, data protection services, sqlpgResSpace, probe:2860
MESSAGE : ADM1823E  The active log is full and is held by application handle
          "274".  Terminate this application by COMMIT, ROLLBACK or FORCE
          APPLICATION.
```

In the preceding example, looking at the DB line, you can see that the WPRCSDB is experiencing full transaction logs.
Another way of viewing the db2diag logs is to log in as the DB2 user and run db2diag:

```
su -l db2inst1
 db2diag | less
```

Related information:

Interpreting diagnostic log file entries

About recovering the messaging subsystem

If the messaging system experiences problems you may need to recover the underlying messaging subsystem.

Typically this involves checking the state of various queues but can also include analyzing the integration bus infrastructure.

Detailed information on recovering the messaging subsystem can be found in the WebSphere Application Server information center.

Related concepts:

"Enterprise service bus messaging infrastructure" on page 165
WebSphere ESB supports the integration of service-oriented, message-oriented, and event-driven technologies to provide a standards-based, messaging infrastructure in an integrated enterprise service bus.

Related information:

Troubleshooting service integration message problems

Using IBM Support Assistant

IBM Support Assistant is a free troubleshooting application that helps you research, analyze, and resolve problems using various support features and tools. IBM Support Assistant enables you to find solutions yourself using the same troubleshooting techniques used by the IBM Support team, and it allows you to organize and transfer your troubleshooting efforts between members of your team or to IBM for further support.

About this task

IBM Support Assistant V4.0 is released with a host of new features and enhancements, making this version the most comprehensive and flexible yet. Our one-stop-shop solution to research, analyze and resolve software issues is now better than ever before, and you can still download it at no charge.

IBM Support Assistant version 4.0 enhancements include:

- **Remote System Troubleshooting:** Explore file systems, run automated data collectors and troubleshooting tools, and view the system inventory on remote systems.
- **Activity-based Workflow:** Choose from support-related activities, or use the Guided Troubleshooter for step-by-step help with analysis and resolution.
- **Case Management:** Organize your troubleshooting data in "cases"; then export and share these cases with other problem analysts or with IBM Support.
- **Improved Flexibility:** Add your own search locations, control updates by hosting your own update site, get the latest product news and updates.

The IBM Support Assistant V4.0 consists of the following three distinct entities:

IBM Support Assistant Workbench

The IBM Support Assistant Workbench, or simply the Workbench, is the client-facing application that you can download and install on your workstation. It enables you to use all the troubleshooting features of the Support Assistant such as Search, Product Information, Data Collection, Managing Service Requests, and Guided Troubleshooting. However, the Workbench can only perform these functions locally, for example, on the system where it is installed (with the exception of the Portable Collector).

If you need to use the IBM Support Assistant features on remote systems, additionally install the Agent Manager and Agent. However, if your problem determination needs are purely on the local system, the Agent and Agent Manager are not required.

The Workbench has a separate download and this is all that is required to get started with the Support Assistant.

IBM Support Assistant Agent

The IBM Support Assistant Agent, or simply the Agent, is the piece of

software that needs to be installed on EVERY system that you need to troubleshoot remotely. Once an Agent is installed on a system, it registers with the Agent Manager and you can use the Workbench to communicate with the Agent and use features such as remote system file transfer, data collections and inventory report generation on the remote machine.

IBM Support Assistant Agent Manager
The IBM Support Assistant Agent Manager, or simply the Agent Manager, needs to be installed only ONCE in your network. The Agent Manager provides a central location where information on all available Agents is stored and acts as the certificate authority. For the remote troubleshooting to work, all Agent and Workbench instances register with this Agent Manager. Any time a Support Assistant Workbench needs to perform remote functions, it authenticates with the Agent Manager and gets a list of the available Agents. After this, the Workbench can communicate directly with the Agents.

The Agent and Agent Manager can be downloaded in a combined installer, separate from the Workbench.

IBM Support Assistant Version 4 has the following functions:

Search interface and access to the latest product information
IBM Support Assistant allows you to search multiple knowledge repositories with one click and gives you quick access to the latest product information so that you spend less time looking for the solution and more time building skills and solving problems.

Troubleshooting tools
Whether you are new to an IBM product or an advanced user, IBM Support Assistant can help. You can choose to be guided through your problem symptoms or view a complete listing of advanced tooling for analyzing everything from logs to memory dumps.

Access to local and remote systems
Using the IBM Support Assistant Workbench installed on a local workstation running the Windows or Linux Intel operating system, you can connect to the IBM Support Assistant Agent installed on a remote system running on the AIX®, Linux, Windows, or Solaris operating system through the IBM Support Assistant Agent Manager on the Workbench. This function enables you to explore, transfer data, and run diagnostic tooling not only on your system but on any other system where the IBM Support Assistant Agent is installed.

Automated data gathering and efficient support
Instead of manually gathering information, you can use IBM Support Assistant to run automated, symptom-specific data collectors. This data can then be attached to an IBM Service Request so that you can get support from the experts at IBM Support.

Procedure

- Follow the installation instructions on IBM Support Assistant (ISA) website at: IBM Support Assistant (ISA).
- Read the “First Steps” section of the documentation for IBM Support Assistant to run the customization wizard, or migrate from a previous version of IBM Support Assistant. Read the “Tutorials” section to learn more about the capabilities of ISA.

Related tasks:

Chapter 26, “Searching knowledge bases,” on page 1123
You can often find solutions to problems by searching IBM knowledge bases. Optimize your results by using available resources, support tools, and search methods.

Chapter 29, “Contacting IBM Software Support,” on page 1131
IBM Software Support provides assistance with product defects.

Related information:

IBM Software support: IBM Support Assistant (ISA)

Chapter 26. Searching knowledge bases

You can often find solutions to problems by searching IBM knowledge bases. Optimize your results by using available resources, support tools, and search methods.

About this task

To search for solutions to your problems in IBM knowledge bases, perform the following steps.

Procedure

1. Search with IBM Support Assistant. IBM Support Assistant (ISA) is a free software serviceability workbench that helps you resolve questions and problems with IBM software products. The ISA tool can search multiple knowledge bases simultaneously

   To search multiple Internet resources for your product, open the ISA and click **Search**. From this page, you can search a variety of resources including:
   - IBM Software Support Documents
   - IBM developerWorks
   - IBM newsgroups and forums
   - Google
   - IBM product information centers

   Note: These free newsgroups and forums do not offer any formal IBM product support. They are intended for user-to-user communication. IBM will not be actively participating in these discussions. However, IBM does review these newsgroups periodically to maintain a free flow of accurate information.
   You may also want to browse the following resources individually.
2. Search with the IBM Software Support Toolbar. IBM Software Support Toolbar is a browser plug-in that provides you with a mechanism to easily search IBM support sites.
3. Search the information center.

   IBM provides extensive documentation in the form of online information centers. An information center can be installed on your local machine or on a local intranet. An information center can also be viewed on the IBM web site. You can use the powerful search function of the information center to query conceptual and reference information and detailed instructions for completing tasks.
4. Search available technical resources. In addition to this information center, the following technical resources are available to help you answer questions and resolve problems:
   - WebSphere ESB technotes
   - WebSphere ESB Authorized Program Analysis Reports (APARs)
   - WebSphere ESB support web site
   - Redbooks Domain
   - IBM Education Assistant
   - WebSphere ESB forums and newsgroups

© Copyright IBM Corp. 2007, 2011

What to do next

Tip:

The following resources describe how to optimize your search results:

- Searching the IBM Support web site
- Using the Google search engine
- IBM Software Support RSS feeds
- My Support e-mail updates

Related tasks:

"Using IBM Support Assistant" on page 1119
IBM Support Assistant is a free troubleshooting application that helps you research, analyze, and resolve problems using various support features and tools. IBM Support Assistant enables you to find solutions yourself using the same troubleshooting techniques used by the IBM Support team, and it allows you to organize and transfer your troubleshooting efforts between members of your team or to IBM for further support.

Related information:

IBM Education Assistant

IBM Software Support Toolbar

Chapter 27. Using IBM Support Assistant

IBM Support Assistant is a free troubleshooting application that helps you research, analyze, and resolve problems using various support features and tools. IBM Support Assistant enables you to find solutions yourself using the same troubleshooting techniques used by the IBM Support team, and it allows you to organize and transfer your troubleshooting efforts between members of your team or to IBM for further support.

About this task

IBM Support Assistant V4.0 is released with a host of new features and enhancements, making this version the most comprehensive and flexible yet. Our one-stop-shop solution to research, analyze and resolve software issues is now better than ever before, and you can still download it at no charge.

IBM Support Assistant version 4.0 enhancements include:

- **Remote System Troubleshooting:** Explore file systems, run automated data collectors and troubleshooting tools, and view the system inventory on remote systems.
- **Activity-based Workflow:** Choose from support-related activities, or use the Guided Troubleshooter for step-by-step help with analysis and resolution.
- **Case Management:** Organize your troubleshooting data in “cases”; then export and share these cases with other problem analysts or with IBM Support.
- **Improved Flexibility:** Add your own search locations, control updates by hosting your own update site, get the latest product news and updates.

The IBM Support Assistant V4.0 consists of the following three distinct entities:

IBM Support Assistant Workbench

The IBM Support Assistant Workbench, or simply the Workbench, is the client-facing application that you can download and install on your workstation. It enables you to use all the troubleshooting features of the Support Assistant such as Search, Product Information, Data Collection, Managing Service Requests, and Guided Troubleshooting. However, the Workbench can only perform these functions locally, for example, on the system where it is installed (with the exception of the Portable Collector).

If you need to use the IBM Support Assistant features on remote systems, additionally install the Agent Manager and Agent. However, if your problem determination needs are purely on the local system, the Agent and Agent Manager are not required.

The Workbench has a separate download and this is all that is required to get started with the Support Assistant.

IBM Support Assistant Agent

The IBM Support Assistant Agent, or simply the Agent, is the piece of software that needs to be installed on EVERY system that you need to troubleshoot remotely. Once an Agent is installed on a system, it registers with the Agent Manager and you can use the Workbench to communicate with the Agent and use features such as remote system file transfer, data collections and inventory report generation on the remote machine.

© IBM Corporation 2004, 2008 © IBM 2007, 2011

IBM Support Assistant Agent Manager
The IBM Support Assistant Agent Manager, or simply the Agent Manager, needs to be installed only ONCE in your network. The Agent Manager provides a central location where information on all available Agents is stored and acts as the certificate authority. For the remote troubleshooting to work, all Agent and Workbench instances register with this Agent Manager. Any time a Support Assistant Workbench needs to perform remote functions, it authenticates with the Agent Manager and gets a list of the available Agents. After this, the Workbench can communicate directly with the Agents.

The Agent and Agent Manager can be downloaded in a combined installer, separate from the Workbench.

IBM Support Assistant Version 4 has the following functions:

Search interface and access to the latest product information
IBM Support Assistant allows you to search multiple knowledge repositories with one click and gives you quick access to the latest product information so that you spend less time looking for the solution and more time building skills and solving problems.

Troubleshooting tools
Whether you are new to an IBM product or an advanced user, IBM Support Assistant can help. You can choose to be guided through your problem symptoms or view a complete listing of advanced tooling for analyzing everything from logs to memory dumps.

Access to local and remote systems
Using the IBM Support Assistant Workbench installed on a local workstation running the Windows or Linux Intel operating system, you can connect to the IBM Support Assistant Agent installed on a remote system running on the AIX, Linux, Windows, or Solaris operating system through the IBM Support Assistant Agent Manager on the Workbench. This function enables you to explore, transfer data, and run diagnostic tooling not only on your system but on any other system where the IBM Support Assistant Agent is installed.

Automated data gathering and efficient support
Instead of manually gathering information, you can use IBM Support Assistant to run automated, symptom-specific data collectors. This data can then be attached to an IBM Service Request so that you can get support from the experts at IBM Support.

Procedure

- Follow the installation instructions on IBM Support Assistant (ISA) website at: IBM Support Assistant (ISA).
- Read the “First Steps” section of the documentation for IBM Support Assistant to run the customization wizard, or migrate from a previous version of IBM Support Assistant. Read the “Tutorials” section to learn more about the capabilities of ISA.

Related tasks:

Chapter 26, “Searching knowledge bases,” on page 1123
You can often find solutions to problems by searching IBM knowledge bases. Optimize your results by using available resources, support tools, and search methods.

Chapter 29, “Contacting IBM Software Support,” on page 1131
IBM Software Support provides assistance with product defects.

Related information:

IBM Software support: IBM Support Assistant (ISA)

Chapter 28. Getting fixes

A product fix might be available to resolve your problem.

About this task

To get product fixes, perform the steps in the Procedure section.

Procedure

1. Determine which fix you need. Check the list of WebSphere ESB recommended fixes to confirm that your software is at the latest maintenance level. Check the list of problems fixed in the IBM WebSphere ESB fix readme documentation that is available for each listed fix pack and refresh pack to see if IBM has already published an individual fix to resolve your problem. To determine what fixes are available using IBM Support Assistant, run a query on `fix` from the search page.

   Individual fixes are published as often as necessary to resolve defects in WebSphere ESB. In addition, two kinds of cumulative collections of fixes, called fix packs and refresh packs, are published periodically for WebSphere ESB, in order to bring users up to the latest maintenance level. You should install these update packages as early as possible in order to prevent problems.

   Note: Fixes specific to the underlying WebSphere Application Server product may also be obtained from the WebSphere Application Server Support Site or from the WebSphere Application Server Support team. Fixes for individual APARs for WebSphere Application Server generally can be applied without affecting WebSphere ESB. However, consult with the software requirements page before updating WebSphere Application Server with cumulative collections of fixes (fix packs). First check to see that the cumulative fix has passed certification, or contact the Support team for verification.
2. Download the fix. Open the download document and follow the link in the **Download package** section. When downloading the file, ensure the name of the maintenance file is not changed. This includes both intentional changes and inadvertent changes caused by certain web browsers or download utilities.
3. Apply the fix. Follow the instructions in the **Installation Instructions** section of the download document. For more information, see the "Updating WebSphere ESB"topic in the "Installing and configuring" documentation.
4. Optional: To receive weekly notification of fixes and updates, subscribe to My Support e-mail updates.

Related tasks:

"Collecting problem reports using IBM Support Assistant Lite" on page 1069
Using IBM Support Assistant Lite (ISA Lite), which is preinstalled in IBM Business Process Manager, you can search for information, investigate problems, or submit a problem report to IBM.

Related information:

Subscribe to My Support e-mail updates

Recommended fixes for WebSphere Enterprise Service Bus

© Copyright IBM Corp. 2007, 2011

Chapter 29. Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

Before you begin

To take advantage of unique Support features, see the WebSphere ESB support page. The Support Page contains the latest information on fixes and downloads, educational resources, and commonly encountered problems and their solutions.

Before contacting IBM Software Support, your company must have an active IBM software subscription and support contract, and you must be authorized to submit problems to IBM. The type of software subscription and support contract that you need depends on the type of product you have. For information about the types of software subscription and support contracts available, see "Enhanced Support" in the *Software Support Handbook* site listed in the Related Topics section.

To contact IBM Software Support with a problem, perform the steps in the Procedure section.

Procedure

1. Define the problem, gather background information, and determine the severity of the problem. For help, see the "Contacting IBM" in the *Software Support Handbook*.
2. Gather diagnostic information. When explaining a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently. For information that IBM Support needs in order to help you solve a problem, see the WebSphere ESB MustGather technote.

   Tip: You can use the WebSphere ESB plug-in for the IBM Support Assistant to capture data and send it to IBM.

   Note: If you are able to determine that the problem is purely with underlying WebSphere Application Server functionality, consider requesting assistance specifically from the WebSphere Application Server Support team rather than the WebSphere ESB team. For information that IBM Support needs in order to help you solve a WebSphere Application Server problem, see the WebSphere Application Server MustGather Technote.
3. Submit your problem to IBM Software Support in one of the following ways:
   - Using IBM Support Assistant: See the "IBM Support Assistant" topic.
   - Online: Open a service request on the IBM Software Support site using the Electronic Service Request (ESR) tool.
   - By telephone: For the telephone number to call in your country or region, go to the contacts page of the IBM Software Support Handbook on the Web and click the name of your geographic region.

What to do next

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis

© Copyright IBM Corp. 2007, 2011

Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround that you can implement until the APAR is resolved. Support will work and communicate with you on the progress and deliver the fix once it is completed. Additionally, once completed, IBM will also publish the resolved APARs on the Software Support Web site, so that other users who experience the same problem can benefit from the same resolution.

Related tasks:

"Collecting problem reports using IBM Support Assistant Lite" on page 1069
Using IBM Support Assistant Lite (ISA Lite), which is preinstalled in IBM Business Process Manager, you can search for information, investigate problems, or submit a problem report to IBM.

"Using IBM Support Assistant" on page 1119
IBM Support Assistant is a free troubleshooting application that helps you research, analyze, and resolve problems using various support features and tools. IBM Support Assistant enables you to find solutions yourself using the same troubleshooting techniques used by the IBM Support team, and it allows you to organize and transfer your troubleshooting efforts between members of your team or to IBM for further support.

Related information:

Software Support Handbook

IBM Software Support site

Chapter 30. Managing applications with WebSphere Application Server

WebSphere ESB is built on WebSphere Application Server, which you can use to deploy and manage WebSphere applications.

About this task

For full information about managing applications with WebSphere Application Server, see the information center and PDF books for WebSphere Application Server Version 7.0, available on the library page at http://www-306.ibm.com/software/webservers/appserv/was/library/v70/.

If you are using the WebSphere ESB documentation plug-ins in your own local information center, you can also add the Application Server documentation plug-ins into the same information center. You can download the documentation plug-ins for WebSphere Application Server Version 7.0 from the library page. For example, WebSphere Application Server Network Deployment (All operating systems) Downloadable information center.

© Copyright IBM Corp. 2007, 2011

Index

A

access control 916, 931
accessibility 28
  keyboard 29
  shortcut keys 29
activation specification 541, 543, 1055, 1056, 1057
adapter 1, 23, 177, 192, 196, 198, 200, 202, 355, 432, 545, 548, 549, 915, 918, 920, 1003
administration task 1059
administrative console 26, 283, 355, 356, 357, 358, 359, 360, 361, 365, 367, 368, 369, 370, 372, 374, 375, 378, 379, 381, 382, 384, 394, 397, 398, 399, 400, 403, 409, 410, 412, 413, 414, 415, 416, 417, 418, 420, 439, 522, 523, 524, 961, 966, 985, 986, 987, 992, 999
administrative interface 355, 356
administrative role 926
administrative security 916, 922, 926
advanced deployment manager profile 283
Aggregation 595
API 609, 620
application bus 168, 169
application security 916, 927
application service 355
application-specific cross-component trace 1071
architecture
  deployment 161
assembly diagram 1101
Asynchronous 633, 636, 640, 688
asynchronous callback 971, 979, 981
asynchronous implementation 971, 976, 981, 983
asynchronous interface 1064
augmentation 283, 1050
authentication 182, 394, 426, 916, 920, 928
authorization 182, 929
Automatic parsing 650

B

bidirectional 30, 32, 33
binding 7, 8, 10, 46, 155, 156, 177, 193, 197, 200, 202, 238, 241, 243, 259, 355, 411, 412, 417, 418, 419, 420, 421, 426, 429, 430, 432, 433, 434, 435, 436, 437, 439, 440, 441, 442, 443, 445, 447, 485, 486, 487, 489, 490, 491, 545, 553, 554, 895, 896
Binding 606, 643, 644, 646, 648, 683, 688, 693, 695
BPM 603
bus 1, 3, 4, 7, 24, 27, 28, 37, 165, 166, 167, 168, 169, 171, 172, 192, 206, 212, 219, 227, 233, 269, 283, 305, 316, 330, 356, 475, 476, 478, 479, 539, 541, 543, 593, 920, 968, 1001, 1003, 1045, 1055, 1057, 1101, 1109, 1110, 1115, 1118, 1129, 1131
Bus 593
bus destination 475, 1101
bus member 539, 541, 543
business data 569, 572, 1081, 1084
business flow 1101
Business Flow Manager
  events
    hold queue 563, 1075
business graph 48
business integration 356, 358, 545, 548, 549
business integration adapter 545, 548, 549
business object 3, 6, 37, 45, 46, 48, 49, 50, 51, 500, 551, 569, 1001, 1002, 1081
Business object 606, 618, 620, 624, 627, 631
business object map 6
business objects
  and WSDL 746
  and XSDs 746
Business Process Choreographer
  events
    failed 563, 1075
    failed, finding 566, 1078
    failed, resubmitting 573, 1085
Business Process Manager 661
business rule 355, 356, 539, 990, 1003, 1004, 1011
business rules manager 356, 539, 990
business space 1, 21, 28, 209, 212, 214, 216, 218, 256, 305, 316, 338, 339, 410, 449, 451, 458, 465, 585
Business Space 661, 662

C

callback 441, 445, 561, 971, 979, 981
Callout 607
Callout 646
Callout node 631, 632, 633, 640
cell 161, 162, 164, 165, 283, 357, 363, 990
client 1, 37, 169, 219, 221, 223, 225, 916, 917, 920
cluster 162, 163, 283, 372, 375, 378, 379, 384, 385, 398, 399, 403, 471, 522, 540, 555, 585, 985, 1064, 1110
cluster member 283, 384, 385, 398, 403
command 163, 283, 356, 362, 365, 369, 370, 373, 374, 375, 376, 377, 381, 382, 383, 384, 388, 389, 391, 392, 393, 399, 401, 408, 418, 420, 426, 430, 433, 436, 439, 443, 476, 483, 484, 485, 486, 487, 489, 490, 491, 496, 525, 526, 539, 540, 541, 543, 560, 924, 929
command assistance 356, 365, 369, 370
common base event 7, 579, 590, 591, 961, 992, 993, 995, 1001, 1059, 1091
common base event browser 992, 993, 995, 1059
common database 283
common event infrastructure 3, 7, 168, 169, 269, 283, 355, 476, 522, 525, 532, 533, 915, 961, 984, 985, 986, 988, 992, 993, 994, 1001, 1059
Common usage patterns 595
component 3, 7, 11, 13, 37, 38, 42, 46, 177, 182, 259, 269, 283, 355, 418, 480, 532, 550, 551, 553, 557, 569, 582, 584, 585, 588, 928, 931, 959, 960, 961, 965, 966, 968, 969, 971, 973, 974, 976, 978, 979, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 992, 993, 995, 997, 999, 1001, 1003, 1013, 1014, 1015, 1031, 1064, 1069, 1071, 1081, 1101, 1110
Component 601, 602, 603, 606, 618, 620, 624, 627, 631, 633, 635, 636, 640
Configuration store 659
connection factory 543
connectivity 177, 192, 227, 436
console 26, 163, 283, 355, 356, 357, 358, 359, 360, 361, 365, 367, 368, 369, 370, 372, 373, 374, 375, 378, 379, 381, 382, 384, 394, 397, 398, 399, 400, 403, 409, 410, 412, 413, 414, 415, 416, 417, 418, 420, 439, 522, 523, 524, 961, 966, 968, 985, 986, 987, 992, 999
Cross-component test 693
cross-component trace 1071
custom deployment environment 384, 385
Custom mediation primitive 614, 624
custom profile 283

D

data binding 46
Data field level 620
data handler 46
Data handler 603
Data Handler mediation primitive 627
data integrity 916, 917
data object 30, 33
data snapshot 1071
data source 165, 167, 283, 384, 394, 396, 397, 403, 971
data store 166, 474
Data transformation 618
database configuration 283, 403

© Copyright IBM Corp. 2007, 2011

database lookup mediation
primitive 253
database provider 396
database script 283
deferred configuration 394
deployment
architecture 161
deployment environment 1, 20, 162, 163, 269, 283, 355, 372, 375, 376, 377, 378, 384, 385, 386, 387, 388, 389, 391, 392, 393, 394, 396, 397, 398, 399, 400, 401, 403, 408, 409, 553, 555, 1109
deployment manager 161, 162, 163, 164, 165, 283, 355, 365, 372, 373, 375, 380, 384, 388, 391, 397, 399, 400, 401, 403, 408, 409, 1059
deployment pattern 1109
deployment target 539
deployment topology 384, 385, 394
diagnostic trace service 985, 986, 987, 988, 992, 993
disability 28
dynamic endpoint 18, 175, 245, 246, 249, 253
dynamic invocation 8
Dynamic invocation 642, 643, 644
Dynamic service gateway pattern 641, 650, 654, 656, 657, 665

E

ejb 430, 432
enabling event sequencing 554, 557, 558, 559
endpoint 8, 18, 19, 173, 175, 181, 189, 206, 208, 209, 218, 219, 223, 225, 229, 232, 233, 238, 245, 246, 249, 251, 253, 269, 316, 330, 433, 435, 439, 440, 442, 443, 447, 465, 466, 467, 469
Endpoint 642, 643, 644, 646, 648
endpoint lookup mediation
primitive 223, 225
Endpoint lookup mediation
primitive 646, 648
enterprise service bus 1, 3, 4, 7, 24, 27, 37, 165, 166, 167, 168, 169, 171, 172, 192, 206, 212, 219, 233, 283, 593, 1001, 1129, 1131
Enterprise Service Bus 593
Error handling 683, 688
event catalog 960, 968, 986, 988, 992, 993, 999, 1001, 1002, 1003
event database 522, 533, 534, 535, 536, 537, 538
event performance statistics 966
event sequencing 550, 551, 553, 554, 555, 556, 557, 558, 559, 561
event service 522, 523, 524, 525, 526
export 7, 10, 38, 42, 177, 192, 193, 259, 401, 413, 417, 418, 420, 421, 426, 430, 432, 433, 434, 435, 436, 437, 439, 440, 441, 442, 443, 447, 449, 471, 483, 489, 490, 491, 553, 554, 557, 558, 1101
Export 603, 620, 624, 627, 631, 642, 643, 644, 646
export binding 7, 10, 193, 417, 418, 420, 426, 430, 432, 433, 434, 435, 436, 439, 442, 443, 447, 489, 490, 553, 554
Export binding 642, 644
export interface 417
exporting deployment environment
definition 384, 400, 403, 409

F

Fail terminal 607
failed event manager 567, 580, 1059, 1062, 1079, 1092, 1101
failed events
finding 566, 1078
managing 563, 1075
resubmitting 573, 1085
First Steps
not starting 1046
Flow Order mediation primitive 614
Function selector 603

G

gate condition 269, 305, 316
generic jms 177, 436, 439, 557
generic JMS 156, 896
governance 172, 181, 227, 232, 269
Governance 648
GTK
warning 1044
guided activity 359

H

high availability 555
hold queue events
managing 563, 1075

I

IBM Business Process Manager 593
IBM Integration Designer 593, 613
identity propagation 916
import 7, 8, 38, 42, 177, 193, 197, 200, 202, 241, 259, 283, 384, 403, 408, 412, 413, 417, 418, 419, 420, 426, 430, 433, 434, 435, 436, 437, 439, 440, 441, 442, 443, 445, 447, 449, 481, 483, 485, 486, 487, 510, 1040, 1064
Import 603, 618, 620, 624, 627, 631, 642, 643, 644, 688, 693, 695
import binding 7, 8, 193, 197, 202, 412, 418, 420, 426, 430, 433, 436, 439, 443, 485, 486, 487
information center 355
informix 536
infrastructure statistics 971
Input terminal 607
installation manager 1040
Integration Designer 601, 603, 607, 618, 620, 624, 627, 640, 641, 642, 643, 644, 683, 688, 691, 693, 695
interface 8, 10, 28, 46, 177, 208, 241, 269, 355, 356, 362, 412, 417, 589, 1015, 1064
Interface 595, 606, 646
invocation pattern 1101

J

jacl 269, 525, 529
Java 609, 610
Java component 603
jca 198, 200, 1055, 1056, 1057
jdbc 167, 198
jms 177, 193, 197, 200, 355, 418, 433, 434, 435, 436, 439, 440, 441, 442, 443, 447, 476, 524, 541, 543, 554, 557, 558, 576, 1088, 1101
JMS 156, 593, 609, 610, 644, 646, 688, 896
events
failed 563, 1075
failed, finding 566, 1078
failed, resubmitting 573, 1085
jndi 430, 434, 439, 441, 445, 527, 529
jython 529, 540, 541

K

keyboard 29
keystore 283

L

launchpad 1045
prerequisites check 1044
Library 606
limitation 155, 156, 895, 896
log analyzer 999

M

managed node 163, 164, 165, 283, 373, 374
managed server 365, 380, 382, 383, 1115
manageprofiles 163, 283, 1042, 1050
Manual parsing 650
mediation 7, 11, 13, 19, 26, 38, 42, 165, 171, 173, 175, 177, 181, 188, 189, 198, 200, 202, 206, 212, 214, 216, 218, 221, 223, 225, 229, 232, 233, 235, 238, 241, 243, 246, 249, 251, 253, 255, 256, 259, 261, 263, 264, 265, 266, 269, 305, 316, 330, 410, 412, 416, 448, 451, 457, 458, 1003, 1013
Mediation 595, 601, 603, 606, 607, 609, 614, 618, 620, 624, 627, 642, 643, 644, 646, 648, 683, 688
mediation flow 11, 42, 165, 173, 188, 229
Mediation flow 603, 607, 614
mediation flow component 11
Mediation flow component 603, 618, 620, 624, 627
mediation module 7, 11, 13, 177, 198, 200, 235, 238, 241, 243, 246, 249, 251, 256, 259, 261, 263, 264, 265, 266, 410, 412, 416
Mediation module 603, 618, 620, 627, 642, 643, 683, 688

mediation policy 19, 212, 214, 216, 218, 255, 256, 269, 305, 316, 448, 451, 457, 458
mediation primitive 7, 11, 13, 173, 175, 181, 221, 223, 225, 229, 235, 246, 249, 251, 253, 261, 263
Mediation primitive 595, 601, 607, 609, 614, 620, 646, 648
Mediation subflow 606
message element setter mediation primitive 261
Message Filter mediation primitive 614
message service client 1, 37, 169
messaging 1, 37, 165, 166, 167, 168, 177, 283, 473, 474, 476, 479, 541, 543, 555, 1109, 1115, 1118
messaging engine 473, 543, 555, 1115
migration 1, 51, 1053
module 1, 7, 8, 10, 11, 13, 18, 19, 24, 26, 37, 38, 51, 173, 175, 177, 189, 198, 200, 208, 209, 212, 214, 216, 218, 229, 235, 238, 241, 243, 245, 246, 249, 251, 253, 255, 256, 259, 261, 263, 264, 265, 266, 269, 305, 316, 330, 355, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 426, 429, 430, 432, 433, 435, 437, 440, 441, 442, 443, 445, 447, 448, 449, 451, 457, 458, 467, 469, 470, 471, 476, 481, 483, 484, 485, 486, 487, 489, 490, 491, 569, 582, 929, 930, 1015, 1055, 1071, 1081, 1101
Module 601, 602, 603, 606, 642, 643, 644, 646, 683, 688
module administration widget 449
MQ 155, 895
  events
    failed 563, 1075
MQ JMS 156, 896

N

network deployment 157, 161, 553, 555
node 161, 162, 163, 164, 165, 283, 372, 373, 374, 380, 384, 385, 391, 392, 403, 408, 471, 990, 1059
Node 607, 631, 632, 633
node agent 161, 162, 164, 165, 372, 373, 374, 380

O

oracle 283, 536, 537
Output terminal 607

P

Patterns Explorer 654
performance 27, 269, 561, 959, 960, 961, 965, 966, 967, 968, 969, 971, 997, 1001, 1015, 1071
policy 19, 182, 212, 214, 216, 218, 255, 256, 269, 305, 316, 420, 421, 448, 451, 457, 458, 922
Policy 646, 648
policy set 420, 421
port 21, 283, 365
Port 646, 648
profile 161, 162, 163, 164, 165, 269, 283, 372, 373, 476, 527, 1035, 1039, 1040, 1042, 1050, 1059
Profile 648
profile management tool 163, 283
Profile Management Tool
  failure 1049
Proxy gateway pattern 650, 654, 658, 659, 660, 662, 664, 665

Q

queue 165, 166, 476, 1101, 1109
queue destination 165, 166

R

Registry lookup 646
reinstallation
  troubleshooting 1048, 1051
relationship manager 157, 160, 269, 497, 516, 517
relationship service 159, 410, 518, 519, 520
remote messaging 283
Request flow 607
resource adapter 1003
Response flow 607
Retry 633
rules manager 356, 539, 990, 1059

S

sample 1, 516, 968, 988, 989, 993, 994
samples 1, 968, 988, 993
samples gallery 968, 988, 993
SCA
  events
    failed 563, 1075
    failed, finding 566, 1078
    failed, resubmitting 573, 1085
schema 13, 514
security 1, 182, 283, 403, 449, 566, 915, 916, 917, 920, 922, 924, 926, 927, 928, 929, 930, 931, 933, 1078
security roles widget 916, 931, 933
service application 7, 24, 410, 411, 412, 413, 476, 483, 485, 489, 561
service component 3, 37, 38, 42, 269, 355, 418, 959, 960, 961, 965, 966, 968, 969, 973, 974, 976, 978, 979, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 992, 993, 995, 997, 999, 1001, 1003, 1015, 1069, 1071, 1110
Service component 601, 602, 603, 606
service component architecture 3, 37, 38, 418, 965, 966, 973, 974, 976, 978, 979, 981, 982, 983, 1015, 1071, 1110
Service Component Architecture 601, 602, 603, 606, 610, 613, 633, 636, 643, 644, 683, 688
Service Data Object 609, 613, 620
service endpoint 19, 175, 330
Service endpoint 646, 648
Service enrichment 620
Service gateway 641, 650, 654, 656, 657, 658, 659, 660, 661, 662, 664, 665
service integration 165, 168, 169, 476, 478, 479, 480, 1101, 1109, 1110, 1115
service integration bus 478, 479, 1101, 1109, 1110, 1115
service invocation 37, 971
Service Invoke mediation primitive 631, 632, 633, 640
service message object 48, 51
Service Message Object 607, 609, 610, 613, 624, 643, 644
Service provider 595, 607, 618, 635, 636, 640
Service requester 595, 607, 618, 635, 636, 640
service-oriented architecture 3, 4, 6, 7, 172
shortcut keys 29
SLA endpoint lookup mediation primitive 648
SMO Attachments 609, 610
SMO Body 609, 610
SMO Context 609, 610
SMO Header 609, 610
soa 3, 37, 171, 172, 173, 177, 181, 182, 187, 189, 232
SOA 601, 602, 603, 606
stand-alone profile 1059
stand-alone server 162, 163, 380, 381
Static endpoint 642, 643
Static service gateway pattern 650, 654, 657, 658, 665
Synchronous 633, 635, 640, 688
system bus 168, 169, 1110

T

topology 384, 385, 387, 394, 539, 540, 1109
trace 569, 571, 574, 591, 985, 986, 987, 988, 992, 993, 999, 1031, 1071, 1081, 1083, 1086
troubleshooting 1070
  reinstallation 1048, 1051
Type Filter mediation primitive 614

U

UDDI 648
ulimit
  warning 1044
Unit test 691
unmanaged node 164, 165

V

Virtual services 660

W

wasservice 283
web service 8, 177, 188, 200, 202, 259, 330, 418, 420, 421, 582, 917, 920

Web Service 643, 644, 646, 648, 691, 693, 695
web service export 420
web service export binding 420
web service import 420
web service import binding 420
Web services test 695
WebSphere Application Server 593
WebSphere ESB 661
WebSphere Message Broker 593
WebSphere MQ 609
WebSphere Transformation Extender 631
widget 21, 206, 209, 214, 269, 305, 316, 330, 338, 339, 410, 448, 449, 451, 458, 585, 916, 927, 931, 933, 1064
wsadmin 26, 362, 365, 369, 370, 375, 376, 377, 384, 388, 389, 391, 392, 399, 401, 408, 484, 485, 486, 487, 489, 490, 491, 496, 520, 525, 526, 539, 540, 541, 543, 926, 990
wsdl 177, 209, 316, 330, 467, 469
WSDL 609, 646, 648, 660, 693, 695
wsrr 13, 19, 181, 189, 206, 208, 214, 216, 218, 219, 221, 223, 225, 232, 246, 255, 256, 269, 305, 316, 355, 410, 457, 492, 493, 494, 495, 496
WSRR 642, 643, 646, 648

X

XPath 613
XSLT 618, 620

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 2007, 2011

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

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

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of WebSphere ESB.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com®, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

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

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

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

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44-1962-816151
  - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

© Copyright IBM Corp. 2007, 2011

SC34-7236-01