webMethods Service Development Help

Version 10.1

October 2017

WEBMETHODS

This document applies to Service Development 10.1 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2008-2022 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at https://softwareag.com/licenses/.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at https://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

Document ID: DES-SD-OLH-101-20220706

Table of Contents

About this Guide **17**
Document Conventions 18
Online Information 18

1 About webMethods Service Development **21**
Before You Use Designer for Service Development 22
Opening the Service Development Perspective 22

2 Working with webMethods Integration Server **25**
Working with Server Definitions 26
Connecting to an Integration Server 33
Disconnecting from an Integration Server 34
Refreshing an Integration Server 34
Notification of Server Shutdown 35
Opening Integration Server Administrator 35
Viewing Integration Server Properties 35
Changing Passwords 35

3 Working with Elements **39**
About Element Names 41
Creating New Elements 41
Guidelines for Working with Elements 43
Opening Elements 43
Closing Elements 44
Editing and Saving Elements 44
Adding Comments for an Element 45
Configuring Dependency Checking for Elements 45
Controlling the Reuse of Elements Published to CentraSite 46
Allowing Editing of Derived Elements 47
Moving and Copying Elements 48
Renaming Elements 52
Refactoring Variable Names in Elements 54
Deleting Elements 61
Finding Dependents and References 62
Inspecting Pipeline References 65
Finding Elements and Variables 67
Filtering Displayed Elements 71
Hiding or Displaying Automatically Generated Flow Services 71
Creating Working Sets 72
Caching Elements 72
Exporting Elements 74
Viewing Server Files for an Element 74
Using Property Templates with Elements 74

4 Assigning and Managing Permissions for Elements....79
What Is an ACL?....80
Assigning ACLs....82
Viewing ACL Information for a Server....84
ACLs and Locking....84
ACLs and Running/Debugging Services....85
ACLs and Creating, Viewing, and Deleting Elements....85
Troubleshooting ACL Usage....85

5 Locking and Unlocking Elements....87
What Is a Lock?....88
About Locking Elements....89
Viewing the Status of Locked Elements....91
Copying, Moving, or Deleting Locked Elements....92
Unlocking Elements....92
Troubleshooting....94

6 Using the Local Service Development Feature....99
About the Local Service Development Workflow....101
How Does Local Service Development Differ from the VCS Integration Feature?....101
Supported Platforms and Eclipse Plug-ins....103
Supported Elements....103
Supported and Unsupported Actions....103
Prerequisites....104
Permissions and Locking....105
Setting the Local Development Server....106
Creating a Local Service Development Project....108
Adding Folders and Elements to the VCS....109
Modifying Packages, Folders, or Elements in the VCS....110
Checking Out an Element from the VCS....111
Checking In Packages and Element to the VCS....112
Getting the Latest Version from the VCS....113
Getting a Specific Version from the VCS....114
Copying Packages from the VCS to Integration Server....115
Reloading a Package....116
Comparing Revisions of an Element....116
Building Java and C Services....117
Deleting a Package Associated with a Local Service Development Project....118
Deleting a Local Service Development Project....118

7 Using the VCS Integration Feature to Check Elements In and Out of a VCS....121
VCS Integration Supported Features....123
VCS Integration Unsupported Features....124
Locking Locally vs VCS Locking....124
System Locking and VCS Integration Feature....124
About Unlocking Elements with Integration Server Administrator....125
Adding New Packages and Elements to a VCS....125

Adding Existing Packages and Elements to a VCS....125
Modifying Elements that are in the VCS....126
Checking Out Packages and Elements....126
Checking In Packages and Elements....127
Reverting Changes to a Checked Out Package or Element....129
Getting the Latest Version from the VCS....130
Getting an Earlier Version from the VCS....130
Deleting Packages and Elements from the VCS....132
Restoring Deleted Items....133
Copying and Moving Folders or Elements....135
Renaming Packages, Folders, and Elements....135
Viewing the History of a Folder or Element....135
Working with Blaze Rules Services....137
Working with Web Service Descriptors....137
Working with webMethods Adapter Connections....137
Working with Java Services....138

8 Managing Packages....141
Creating a Package....142
Documenting a Package....143
Viewing Package Settings, Version Number, and Patch History....144
Assigning a Version Number to a Package....145
About Copying Packages Between Servers....145
Reloading a Package....147
Comparing Packages....148
Deleting a Package....148
Exporting a Package....148
About Package Dependencies....149
Assigning Startup, Shutdown, and Replication Services to a Package....151

9 Building Services....157
A Process Overview....158
Package and Folder Requirements....159
About the Service Signature....159
About Service Run-Time Parameters....165
About Automatic Service Retry....178
About Service Auditing....180
Using a Circuit Breaker with a Service....187
About Universal Names for Services or Document Types....191
About Service Output Templates....195
Printing a Flow Service....197
Comparing Flow Services....198

10 Building Flow Services....199
What Is a Flow Service?....200
Building Services Using the Tree Tab or Layout Tab....203
Creating a New Flow Service....203
Setting Properties for a Flow Step....215

The INVOKE Step....215
The BRANCH Step....218
The REPEAT Step....226
The SEQUENCE Step....232
The LOOP Step....234
The EXIT Step....239
The MAP Step....241

11 Working in the Layout Tab....243
What Is the Layout Tab....244
What Does a Flow Service Look Like in the Layout Tab?....244
Show or Hide the Grid Lines in the Flow Service Editor....247
Building a Flow Service in the Layout Tab....247

12 Mapping Data in Flow Services....251
What Does the Pipeline View Contain?....252
Basic Mapping Tasks....256
About Linking Variables....256
About Assigning Values to Pipeline Variables....271
Dropping Variables from the Pipeline....275
Adding Variables to the Pipeline....276
Working with Transformers....278
Testing Flow Steps Before Running a Flow Service....285
Mapping Using ForEach....286

13 Performing Data Validation....293
Blueprints or Models Against which Data is Validated....294
Performing Input/Output Validation....295
Performing Pipeline Validation....297
Performing Document Validation....297
Performing XML Validation in Integration Server....298
Performing Validation from within a Java Service....299
Validation Errors....300
Validation Exceptions....300
Preventing Running Out of Memory Error During Validation....301

14 Building Java Services....303
Overview of Building Java Services....304
Java Service Editor....305
Service Development Projects in the Local Workspace....308
How Java Services Are Organized on Integration Server....309
Creating a Java Service....310
Using an IData Object for the Java Service Input and Output....312
Generating Java Code from Service Input and Output Parameters....314
Editing an Existing Java Service....316
Adding Classes to the Service Development Project....317
Compiling a Java Service....319
Generating Code a Java Service Can Use to Invoke a Specified Service....320

Deleting a Java Service....322

15 Building Java Services in Your Own IDE....323
How Java Services are Organized on Integration Server....324
Requirements for the Java Service Source Code....326
IData Object for Java Service Input and Output....326
Adding Comments to Your Java Code for the jcode Utility....327
Using the jcode Utility....331

16 Building Map Services....337
What Is a Map Service?....338
Building Map Services Using the Tree Tab or Graphical View Tab....338
Creating a Map Service....340
Setting Properties for a Map Service....341
Working in the Graphical View tab....341
Debugging Map Services....342

17 Building C/C++ Services....343
The Java Code for a C/C++ Service....344
Overview of Building C/C++ Services....344
Prerequisites for Building C/C++ Services....345
C/C++ Service Editor....346
Service Development Projects in the Local Workspace....349
How C/C++ Services Are Organized on Integration Server....351
Creating a C/C++ Service....351
Generating C/C++ Code from Service Input and Output Parameters....354
Adding Classes to the Service Development Project....355
Building the C/C++ Source Code....356
Compiling the C/C++ Source Code....357
Generating Code a C/C++ Service Can Use to Invoke a Specified Service....359
Debugging C/C++ Services....360

18 Building Services from .NET Methods....363
Environment Setup for Creating .NET Services....364
.NET Service Editor....364
Creating a .NET Service....366
Modifying the .NET Assembly Information....367
Modifying the Class Lifetime for a .NET Service....368
Running a .NET Service in Designer....369

19 Building XSLT Services....371
What Is XSLT?....372
What Is an XSLT Service?....372
How Does an XSLT Service Work?....373
What Is a Translet?....373
About the XSLT Service Editor....374
Overview of Building XSLT Services....374

Creating an XSLT Service....374
XSLT Service Signature....375
Running an XSLT Service....378
Debugging an XSLT Service....378
Guidelines for the XSLT Style Sheet....379
Using Name/Value Pairs with an XSLT Service....380
Configuring XSLT Transformer Factory Settings....386

20 Managing Cloud Connector Services....387
Creating a Cloud Connector Service....388
Editing a Cloud Connector Service for a SOAP-Based Provider....389
Editing a Cloud Connector Service for a REST-Based Provider....394

21 Running Services....403
Using Launch Configurations to Run Services....404
Supplying Input Values to a Service....406
Running a Service....421
Viewing Results from Running a Service....422
Running Services from a Browser....426

22 Debugging Flow Services....429
About Debugging Flow Services....430
Creating Launch Configurations for Debugging Flow Services....432
Debugging a Flow Service....434
Stepping Through Flow Services....435
Using Breakpoints When Debugging Flow Services....439
Disabling and Enabling Flow Steps and Transformers....442
Disabling and Enabling Conditions....443
Modifying the Flow Service Pipeline while Debugging....444
Saving and Restoring the Flow Service Pipeline while Debugging....447
Viewing Service Results from a Flow Service Debug Session....449
Using the Server Log for Debugging....449

23 Debugging Java Services....455
About Debugging a Java Service while its Class Runs in Designer....456
About Test Harnesses....457
About Java Application Launch Configuration....459
How to Suspend Execution of a Java Class while Debugging....462
Debugging a Java Service while its Class Runs in Designer....462
About Debugging a Java Service while it Runs in Integration Server....465

24 Working with REST....473
Approaches for Developing REST Resources....474
Generating a REST Resource Using the Legacy Approach....474
About the Folder for REST Resources Created Using the Legacy Approach....476
Configuring a REST Resource Using the URL Template-Based Approach....476

25 Working with REST API Descriptors....481
Overview of Creating a REST API Descriptor....482
Creating a REST API Descriptor from a REST Resources....483
Editing General Information for a REST API Descriptor....485
Changing the Available MIME Types for a REST API Descriptor....486
Working with REST Resources in a REST API Descriptor....486
Viewing REST V2 Resources in Group by Tags Mode....494
Running a REST V2 Resource Operation from REST API Descriptor Editor....495
About REST Definitions....495
Viewing the Swagger Document for a REST API Descriptor....496
Mapping Integration Server Data Types to Swagger Data Types....496
Creating a REST API Descriptor from a Swagger Document....497
Publishing REST API Descriptors to API Portal....500

26 Working with OData Services....503
Understanding OData Service Terminology....504
Supported and Unsupported OData Features....505
Overview of Creating an OData Service....506
Creating an OData Service....506
Adding OData Elements to the OData Service....509
Adding Properties to the OData Elements....510
Adding Associations to OData Elements....511
Editing the OData Service....511
Synchronizing the External Entity Type....512
How Integration Server Processes an OData Service Request....512
Querying Data Using $filter....513

27 Working with Document Types....515
Creating an IS Document Type....516
Editing Document Types....551
About Universal Names and Document Types....552
Printing an IS Document Type....552
Working with Publishable Document Types....553
Deleting Publishable Document Types....574
About Testing Publishable Document Types....575
About Synchronizing Publishable Document Types....580
Publishing Documents as JMS Messages....589

28 Working with XML Document Types....593
What Is an XML Document Type?....594
Why Use XML Document Types Instead of IS Document Types?....594
Creating an XML Document Type....597

29 Working with Specifications....599
Creating a Specification....600

30 Working with Variables....603
Creating a Document Reference or a Document Reference List Variable....604
Using XML Namespaces and Namespace Prefixes with Variables....605
Assigning Display Types to String Variables....607
About Variable Constraints....607

31 Working with Schemas....615
What Does an IS Schema Look Like?....616
Creating an IS Schema....621
About Editing Simple Type Definitions....625
About Schema Domains....627

32 Working with JMS Triggers....629
About SOAP-JMS Triggers....630
Overview of Building a Non-Transacted JMS Trigger....631
Standard JMS Trigger Service Requirements....632
Creating a JMS Trigger....633
Managing Destinations and Durable Subscribers on the JMS Provider through Designer....642
Building Standard JMS Triggers with Multiple Routing Rules....643
Enabling or Disabling a JMS Trigger....644
Setting an Acknowledgement Mode....646
About Join Time-Outs....647
About Execution Users for JMS Triggers....649
About Message Processing....649
Fatal Error Handling for Non-Transacted JMS Triggers....656
Transient Error Handling for Non-Transacted JMS Triggers....658
Exactly-Once Processing for JMS Triggers....664
Debugging a JMS Trigger....666
Building a Transacted JMS Trigger....668

33 Working with webMethods Messaging Triggers....679
Overview of Building a webMethods Messaging Trigger....680
Creating a webMethods Messaging Trigger....682
Disabling and Enabling a webMethods Messaging Trigger....695
About Join Time-Outs....696
About Priority Message Processing....698
About Execution Users for webMethods Messaging Triggers....700
About Capacity and Refill Level for the webMethods Messaging Trigger Queue....701
About Document Acknowledgements for a webMethods Messaging Trigger....704
About Message Processing....705
Fatal Error Handling for a webMethods Messaging Trigger....714
About Transient Error Handling for a webMethods Messaging Trigger....716
Exactly-Once Processing for webMethods Messaging Triggers....723
Modifying a webMethods Messaging Trigger....726
Deleting webMethods Messaging Triggers....727
Running a webMethods Messaging Trigger with a Launch Configuration....729
Debugging a webMethods Messaging Trigger....734

34 Transient Error Handling During Trigger Preprocessing....737
Server and Trigger Properties that Affect Transient Error Handling During Trigger Preprocessing....738
Overview of Transient Error Handling During Trigger Preprocessing....739

35 How Triggers Correspond to Durable Subscribers on Universal Messaging....741

36 Working with Web Services....743
What Are Web Service Descriptors?....745
About Provider Web Service Descriptors....746
About Consumer Web Service Descriptors....760
About Refreshing a Web Service Descriptor....771
Viewing the WSDL Document for a Web Service Descriptor....781
WS-I Compliance for Web Service Descriptors....782
Changing the Target Namespace for a Web Service Descriptor....784
Viewing the Namespaces Used within a WSDL Document....784
Enabling MTOM/XOP Support for a Web Service Descriptor....784
Adding SOAP Headers to the Pipeline....785
Validating SOAP Response....786
Validating Schemas Associated with a Web Service Descriptor....787
Omitting xsd:any from the WSDL Document....789
Working with Binders....790
Working with Operations....799
Adding Headers to an Operation....805
About SOAP Fault Processing....807
Viewing Document Types for a Header or Fault Element....813
Working with Handlers....813
Working with Policies....816
About Pre-8.2 Compatibility Mode....818

37 Working with UDDI Registry....825
Opening UDDI Registry View....826
Connecting to a UDDI Registry....827
Disconnecting from a UDDI Registry....828
Refreshing a UDDI Registry Session....828
Browsing for Web Services in a UDDI Registry....828
Creating a Web Service Descriptor from a UDDI Registry....829
Publishing a Service to UDDI Registry....830
Deleting a Service from UDDI Registry....831

38 Working with Flat Files....833
Concepts....834
Creating Flat File Schemas....835
Testing Flat File Schemas....849
Creating Flat File Dictionaries....852
Defining Flat File Elements....855

39 Working with Adapters....861
About Adapter Connections....862
About Adapter Services....862
About Adapter Listeners....862
About Adapter Notifications....863

40 Subscribing to Events....865
What Happens When an Event Occurs?....867
Subscribing to Events....868
Viewing and Editing Event Subscriptions....873
Suspending Event Subscriptions....873
Deleting an Event Subscription....874
Building an Event Handler....874
Invoking Event Handlers Synchronously or Asynchronously....875
About Alarm Events....875
About Audit Events....876
About Audit Error Events....876
About Exception Events....876
About Guaranteed Delivery Events....877
About JMS Delivery Failure Events....878
About JMS Retrieval Failure Events....878
About Port Status Events....879
About Replication Events....879
About Security Events....880
About Session Events....881
About Stat Events....881
About Transaction Events....881

41 Submitting and Receiving XML Documents....883
Submitting and Receiving XML in a String Variable....885
Submitting and Receiving XML in $xmldata....886
Submitting and Receiving XML via HTTP....888
Submitting and Receiving XML via FTP....893
Submitting and Receiving XML via E-mail....896

42 Working with Load and Query Services....899
What Are the Load and Query Services?....900
Basic Concepts....900
About the pub.xml:loadXMLNode Service....901
About the pub.xml:loadEnhancedXMLNode Service....901
About the pub.xml:queryXMLNode Service....901

43 Building Services that Retry....903
Requirements for Retrying a Service....904
Example Service that Throws an Exception for Retry....905

44 Creating Client Code....909
Building a Java Client....910
Building a C/C++ Client....912
Building a Browser-Based Client....913
Building a REST Client....919

45 Comparing Integration Server Packages and Elements....921
Working with the Compare Editor....922
Comparing Packages and Elements....925

46 Connecting to webMethods API Portal for Publishing REST API Descriptors....929
Configuring a Connection to API Portal....930

47 API Portal Preferences....935

48 Document Expansion Preferences....937

49 Integration Server Preferences....939

50 Service Development Preferences....941
Adapter Service/Notification Editor Preferences....942
Compare Editor Preferences....942
Element Property Templates Preferences....943
Flow Service Editor Preferences....944
HTML Generation Preferences....945
Java/C Service Editors Preferences....945
Launching Preferences....946
Local Service Development Preferences....946
Package Navigator Preferences....947
Publishable Document Type Preferences....949
Results View Preferences....949
Run/Debug Preferences....949
Schema Editor Preferences....950
Swagger Syntax Coloring Preferences....950
Web Service Descriptor Editor Preferences....951

51 Properties....953
Integration Server Properties....955
Package Properties....957
Element Properties....962
Document Type Properties....964
Flat File Dictionary Properties....970
Flat File Element Properties....970
Flat File Schema Properties....984
JMS Trigger Properties....987

Link Properties....997
OData Service Properties....998
REST V2 Resource Properties....1003
REST API Descriptor Properties....1003
Schema Properties....1006
Schema Component Properties....1007
Service Properties....1022
Specification Properties....1036
Transformer Properties....1037
Variable Properties....1038
Web Service Connector Properties....1042
Web Service Descriptor Properties....1047
webMethods Messaging Trigger Properties....1060
WebSocket Endpoint Properties....1067

52 webMethods Flow Steps....1069
BRANCH....1070
EXIT....1073
INVOKE....1074
LOOP....1075
MAP....1077
REPEAT....1078
SEQUENCE....1080

53 Data Types....1083
Data Types in IData Objects....1084
Java Classes for Objects....1085
How Designer Supports Tables....1087

54 Icons....1089
Package Navigator View Icons....1090
UDDI Registry View Icons....1093
Flat File Element Icons....1093
Flow Step Icons....1094
OData Service Icons....1095
REST API Descriptor Icons....1096
Schema Component Icons....1096

55 Toolbars....1101
Compare Editor Toolbar....1102
Document Type Editor Toolbar....1102
Flat File Schema and Dictionary Editors Toolbars....1103
Package Navigator View Toolbar....1103
Pipeline View Toolbar....1103
REST API Descriptor Toolbar....1105
Service Editor Toolbar....1105
Results View Toolbar....1106
Specification Editor Toolbar....1107

UDDI Registry View Toolbar....1107
Variables View Toolbar....1108
Web Service Descriptor Editor Toolbar....1108

56 Keyboard Shortcuts....1111

57 Conditional Expressions....1113
Guidelines for Writing Expressions and Filters....1114
Syntax....1114
Operators for Use in Conditional Expressions....1117
Operator Precedence in Conditional Expressions....1123
Addressing Variables....1124
Rules for Use of Expression Syntax with the Broker....1126

58 Regular Expressions....1129
Using a Regular Expression in a Mask....1130
Regular Expression Operators....1130

59 Validation Content Constraints....1137
Content Types....1138
Constraining Facets....1148

60 webMethods Query Language....1151
Overview....1152
Object References....1152
Sibling Operators....1153
Object Properties....1155
Property Masking....1155

About this Guide

- Document Conventions 18
- Online Information 18

webMethods Service Development provides tools and features that developers can use to build and test services. webMethods Service Development also provides tool to connect to Integration Server, manage packages, and create the elements needed to support services such as document types, triggers, and web service descriptors. You can learn more by looking in Contents for **Software AG Products > webMethods Service Development Help**.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service*.
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 About webMethods Service Development

- Before You Use Designer for Service Development 22
- Opening the Service Development Perspective 22

Software AG Designer provides a set of Service Development features that you can use to build, edit, and debug services and integration logic. It provides a collection of editors and views in which you can develop the logic and supporting objects (referred to as *elements*) for an integration solution. It also provides tools for running and debugging the solutions you create.

Designer lets you rapidly construct integration logic with an easy-to-use implementation language called the *webMethods flow language*. Flow language provides a set of simple but powerful constructs that you use to specify a sequence of actions (steps) that the Integration Server will execute at run time. Designer also has extensive data transformation and mapping capabilities that allow you to quickly drag-and-drop data fields from one step to the next.

Besides providing tools for constructing flow services, Designer provides additional editors and tools for creating various elements that support the execution of an integration solution. For example, you use Designer to create the document types and schemas used for data validation and to define triggers that launch the execution of services when certain messages are received.

Before You Use Designer for Service Development

Designer builds and edits services and other elements *directly* on a webMethods Integration Server. To use Designer for service development, you must

- Have access to an Integration Server on which you can build and debug services.
- Have a user account on that webMethods Integration Server.
- Belong to a group that is a member of the "Developers" ACL (access control list) on that Integration Server.
- Create a server definition that defines the connection between Designer and Integration Server

If you do not have access to a Integration Server or you do not have an appropriate user account or access rights, see your server administrator.

Note:
This guide describes features and functionality that may or may not be available with your licensed version of webMethods Integration Server. For information about the licensed components for your installation, see the **Settings > License** page in the webMethods Integration Server Administrator.

Opening the Service Development Perspective

Software AG Designer provides a Service Development perspective that contains the views, editors, and tabs needed to build services and other supporting elements.

To open the Service Development perspective

1. In Designer, select **Window > Open Perspective > Other**.
2. In the **Open Perspective** dialog box, select **Service Development**. Click **OK**.

Designer switches to the Service Development perspective.

2 Working with webMethods Integration Server

- Working with Server Definitions 26
- Connecting to an Integration Server 33
- Disconnecting from an Integration Server 34
- Refreshing an Integration Server 34
- Notification of Server Shutdown 35
- Opening Integration Server Administrator 35
- Viewing Integration Server Properties 35
- Changing Passwords 35

webMethods Integration Server provides an environment for the orderly, efficient, and secure, execution of services. It decodes client requests, identifies the requested services, invokes the services, passes data to them in the expected format, encodes the output produced by the services, and returns output to the clients.

Using Designer, you build and edit services, document types, and other elements *directly* on an Integration Server. You connect Designer to Integration Server through *server definitions*. A server definition specifies the location and characteristics of the Integration Server to which Designer is connecting.

Note: Designer can connect to the equivalent or previous versions of Integration Server only. For example, Designer version 10.1 can connect to Integration Server version 10.1 or any previous versions of Integration Server back to and including version 9.5. However, Designer cannot connect to a higher version of Integration Server. For example, Designer 9.9 cannot connect to Integration Server 10.1.

Working with Server Definitions

A server definition specifies the location and characteristics of an Integration Server to which Designer connects. You connect to Designer through server definitions. You create and view server definitions on the **Window > Preferences > Software AG > Integration Server s** page.

Creating Server Definitions

By default, a new Designer installation includes a single server definition named Default. This server is marked as the default server and is configured to use `localhost:5555`.

If your Designer installation needs to connect to more Integration Servers, you can create additional server definitions.

To create a server definition

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Click **Add**.
4. In the **Add Integration Server** dialog box, enter the following information:

Field	Description
Name	Name to use for this Integration Server. **Note:**

Field	Description
	The name cannot contain control characters, special characters, and characters outside of the basic ASCII character set, such as multi-byte characters.
Host	Host name (for example, `workstation5.webmethods.com`) or IP address (for example, `132.906.19.22` or `2001:db8:85a3:8d3:1319:8a2e:370:7348`) of the Integration Server to connect to. The host names or IP addresses can include upper and lower case alphabetic characters, digits (0-9), hyphens (-), and periods (.) but cannot include spaces. For IPv6, IP addresses can also include colons (:) and brackets ([]).
Port	Port number to connect to on the Integration Server.
User	Indicates the name of a valid user account on the Integration Server. The user name must be a member of a group belonging to the Developers ACL. Use the exact combination of upper- and lower-case characters with which it was originally defined. Integration Server user names are case sensitive.
Password	The password for **user**. Use the exact combination of upper- and lower-case characters with which it was originally defined. Integration Server passwords are case sensitive.
Connect immediately	Indicates whether Designer should connect to the Integration Server immediately after you add or edit a definition and click **OK** on the Add Integration Server or Edit Integration Server dialog box.
Connect at startup	Indicates whether Designer should automatically connect to the Integration Server at Designer startup.
Secure Connection	Indicates whether the session will be opened through HTTP or HTTPS. If you want to open an HTTPS session on the selected server using the Secure Socket Layer (SSL), select this check box. If you want to open an HTTP session on the server, clear this check box.

Note:
If you want to setup multiple connections to the same Integration Server (same IP address and same port) using different hostnames then you must deselect the **Verify hostname for server connections** check box under **Window > Preferences >Software AG > Service Development> Package Navigator**.

5. To test this connection, click **Verify Server**.

6. Click **OK**.

By default, Designer will automatically connect to the Integration Server. If Designer does not automatically connect to the server, click **Connect** on the **Preferences** page.

Fetching Server Definitions from an Integration Server

You can obtain a server definition by *fetching* it from another Integration Server. Keep the following points in mind when fetching server definitions:

- For the fetch process to work, the WmDesigner package must be installed and running on the other Integration Server, and the server definition you are trying to fetch must have been previously defined on the copy of Designer running on the other server.
- If the Integration Server from which you want to fetch definitions contains remote server definitions, make sure that the remote server definitions use host names or addresses that can be resolved by the machine retrieving the definitions.

To fetch to an Integration Server definition from another Integration Server

1. In Designer, select **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Click **Fetch**.
4. In the **Fetch** dialog box, enter the following information about the Integration Server from which you want to fetch a server definition.

Field	Description
Host Name or IP Address	Host name (for example, `workstation5.webmethods.com`) or IP address (for example, `132.906.19.22` or `2001:db8:85a3:8d3:1319:8a2e:370:7348`) of the Integration Server to connect to. The host names or IP addresses can include upper and lower case alphabetic characters, digits (0-9), hyphens (-), and periods (.) but cannot include spaces. For IPv6, IP addresses can also include colons (:) and brackets ([]).
Port	Port number on which Integration Server listens for requests.
Secure Connection	Indicates whether to connect to Integration Server through an HTTP or HTTPS connection. Select the check box to connect through an HTTPS connection.
User	The name of a valid user account on this server. Use the exact combination of upper- and lower-case characters with which it was originally defined. Integration Server user names are case sensitive.

Field	Description
	Note: The server is installed with a default user account called “Developer” that has developer privileges.
Password	The password for the user account in **User**. Use the exact combination of upper- and lower-case characters with which it was originally defined. Integration Server passwords are case sensitive.

5. Click **Connect.**

   Designer populates the bottom half of the **Fetch Integration Server Definitions** dialog with a list of server definitions available on the other Integration Server.

6. Select one or more definitions to fetch, and click **OK.**

   Designer refreshes the **Integration Server s** page, this time including the fetched definitions. Designer automatically tries to connect to the fetched servers.

7. For any server definitions that have the status **No user or password**, select the definition, click **Edit**, supply the user ID and password, and click **OK**.

Importing Server Definitions

You can import server definitions from a properties file. A properties file can contain one or more server definitions.

The default properties file name is logicalServer.properties. It resides in the top level of the Designer folder in the following directory: <*workspace_location*>\.metadata\.plugins\com.softwareag.is.core. The file contains the initial installed server definition, which specifies a host name and port of localhost:5555, is named Default, and is marked as the default server.

You can use properties files to make existing server definitions available to other Designer users in cases where it is not possible to fetch server definitions.

When you import servers definitions, you overwrite all existing definitions in your workspace.

To import a server definition properties file

1. In Designer: **Window > Preferences**

2. In the preferences navigation tree, select **Software AG > Integration Server s**.

3. Click **Import**.

4. In the **Open** window, navigate to the .properties file you want to import.

5. Click **OK** to import the data from the selected .properties file into your **Preferences > Software AG > Integration Server s** screen.

Exporting Server Definitions

You can export server definitions to a properties file. When you export server definitions, you export all definitions in your workspace.

You can save the file with any name and in any location, but the file must be saved with a .properties extension.

To export server definitions to a properties file

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Click **Export**.
4. In the **Save As** dialog box, navigate to the folder where you want to save the server definition you are exporting, and type a file name. You do not have to type the .properties extension.

   You can also click an existing .properties file if you want to overwrite it with the new definitions.
5. Click **OK** to save the .properties file.

Removing Server Definitions

To remove a server definition

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Select the server definition you want to remove.
4. Click **Remove**.

   Designer prompts you to confirm that you want to remove this server definition. If this server definition is the default, Designer will remind you that you need to define a new default after this server definition has been deleted.
5. If the server definition you deleted was the default, you need to define a new default. For instructions on defining a default server definition, see "Setting a Default Server Definition" on page 31.

Editing Server Definitions

You can change the properties of a server definition by editing it.

Sometimes there are changes on the associated Integration Server that require you to update the server definition. For example, if the port number changes, you must update the server definition to reflect that change.

Other times, you might decide to change the name of the server definition, or to change whether Designer automatically connects to this Integration Server at Designer startup or when the definition is updated.

If a server definition displays a status of **No userid or password**, you can edit the definition to add the user and password.

For a detailed description of the fields you can change, see " Integration Server Preferences" on page 939.

To edit a server definition

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Select the server definition that you want to edit.
4. Click **Edit**.
5. Enter new values in the fields you want to change.
6. In the **Edit Integration Server** dialog, click **OK**.
7. In the **Preferences** page, click **OK**.

Considerations for Process Development

If you will be working in Process Development, you might want to make the following changes your server definitions to:

- Specify a server definition as the default.
- Place server definitions offline.
- Bring server definitions online.

Setting a Default Server Definition

By default, a new Designer installation includes a server definition named Default. This server is marked as the default server and is configured to use *localhost:5555*. If a user creates or edits a process and no server definitions are connected, Designer automatically connects to the default server definition.

If you update the configuration so that a different server definition is the default, and a user subsequently creates a step when Designer is not connected to an Integration Server, Designer will use the new default server for the new steps. In contrast, Designer will continue to use the original servers for existing steps.

There must be one and only one default Integration Server defined at all times.

To set a default server definition

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Check the **Default** box for the server definition you want to be the default.

   Designer prompts you to verify that you want to replace the existing default with the new one.
4. Click **OK**.

Placing a Server Definition Offline

Typically, when you work with Designer, it is connected to the Integration Servers defined by the server definitions on the Integration Servers Preferences page. Process Development users, however, can perform tasks without being connected to an Integration Server. By placing a server definition *offline*, you can prevent Designer from trying to connect to the associated server and prompting you for credentials.

When you select the **Offline** check box for a server definition, Designer terminates any existing connections to the host:port combination specified in this definition. For example, if server definition A and server definition B both specify localhost:5555, selecting the **Offline** check box to the right of A will terminate the server connections for both A and B.

Refer to the *webMethods BPM Process Development Help* for more information.

To place a server definition in offline status

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Select the **Offline** check box to the right of server definition you want to take offline.

4. Click **OK**.

Bringing a Server Definition Online

Designer must be connected to an Integration Server so that you can view and update Integration Server assets such as flow services, document types, JMS triggers, and web service descriptors. There may be times, however, when you find that a server definition that you need to work with is offline.

To bring a server definition online

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Clear the **Offline** check box to the right of server definition you want to bring online.
4. If Designer issues a message stating that you must enter a user name and password, click the **Edit** button and provide the user name and password.
5. Click **OK**.

Connecting to an Integration Server

When you connect to an Integration Server, you create a session on that Integration Server. You maintain a session on that server until you exit Designer or disconnect from the server. You can have open sessions on multiple servers at a time.

To connect to an Integration Server

1. In Package Navigator view, select the server to which you want to connect.
2. Right click, and select **Connect to server**.

Connecting to an Integration Server through Preferences

Designer must be connected to an Integration Server so that you can view and update Integration Server assets such as flow services, document types, JMS triggers, and web service descriptors.

To connect to an Integration Server

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.

3. On the Integration Servers page, select the server to which you want to connect and click **Connect**.

Disconnecting from an Integration Server

While you have an open session on a server through Designer, you are using a licensed seat for that server. At times when you are not actively using Designer, you may want to close your session to free a seat on the server for others to use.

To disconnect from an Integration Server

1. In Package Navigator view, select the Integration Server from which you want to disconnect.

2. Right click, and select **Disconnect from server**.

Disconnecting from an Integration Server via Preferences

While you have an open session on a server through Designer, you are using a licensed seat for that server. At times when you are not actively using Designer, you may want to close your session to free a seat on the server for others to use.

To disconnect from an Integration Server

1. In Designer: **Window > Preferences**

2. In the preferences navigation tree, select **Software AG > Integration Server s**.

3. On the Integration Servers page, select the server from which you want to disconnect and click **Disconnect**.

Refreshing an Integration Server

Package Navigator view is not dynamically updated when other users lock, unlock, add, delete, or rename elements on a server. You can refresh an Integration Server to reflect any changes made to the contents of the servers with which you are working.

To refresh an Integration Server

1. In Package Navigator view, select the Integration Server you want to refresh.

2. Right click, and select **Refresh**.

Notification of Server Shutdown

If the server administrator shuts down the server on which you have an open session, Designer does one of the following:

- If the server administrator specified a time delay before shutdown, Designer displays a message notifying you when the shutdown process began and how many minutes remain before the server shuts down. After you receive notification of server shutdown, save any work that you want to keep and then close your session. If you do not close your session, Designer notifies you when the server has shut down.
- If the server administrator performed an immediate shutdown, Designer displays a message stating that your connection to the server has been lost. (Designer also displays this message if the network connection to the server is lost.)

Designer restores your connection to the server when the server restarts. If you did not save your work before shut down occurred, Designer prompts you to save your work after the connection is re-established.

Opening Integration Server Administrator

There may be times when you need to perform administrator tasks on an Integration Server.

To open Integration Server Administrator

- In Package Navigator view, right-click the Integration Server for which you want to open Integration Server Administrator and select **Open Administration View**.

Viewing Integration Server Properties

- In Package Navigator view, right-click the Integration Server for which you want to open Integration Server Administrator and select **Properties**.

  Designer displays the Properties screen, from which you can display information about the following areas of the Integration Server: Event Manager, Locked Elements, Server ACLs, and Server Information. Refer to " Integration Server Properties" on page 955 for a description of these properties.

Changing Passwords

You can change the password for your user account. If you forget your password, contact the server administrator.

Important:
You cannot use Designer to change passwords of users that are stored in an external directory. For information about managing users stored in an external directory, see *webMethods Integration Server Administrator's Guide*.

Password Requirements

For security purposes, Integration Server places length and character restrictions on passwords. Integration Server contains a default set of password requirements; however, your server administrator can change these. For more information about these password requirements, contact your server administrator.

The default password requirements provided by webMethods Integration Server are as follows:

Requirement	Default
Minimum length	8
Minimum number of alphabetic characters	3
Minimum number of uppercase characters	2
Minimum number of lowercase characters	2
Minimum number of numeric characters	1
Minimum number of special characters (non-alphabetic and non-numeric characters, such as *. ?,)	1

Note:
The use of special characters is regulated by the following restrictions:

- A password cannot begin with an asterisk (*).
- Passwords cannot contain quotation marks ("), backslashes (\), ampersands (&), or less-than signs (<). Use the watt.server.illegalUserChars configuration property to restrict the use of additional characters.

To ensure the security of your password, follow the additional guidelines below:

- Do not choose obvious passwords, such as your name, address, phone number, license plate, name of your spouse or child, or a birthday
- Do not use any word that can be found in the dictionary.
- Do not write your password down.
- Do not share your password with anyone.
- Change your password frequently.

Changing Your Password

Note:
If you are outside of the corporate firewall, do not change your password unless you use SSL to open the session on the Integration Server. If you do not use SSL, your password can be exposed in unencrypted form.

To change your password

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Integration Server s**.
3. Select the server definition for which you want to change the password and click **Change Password**.
4. In the Change Password dialog box, in the **Old password** field, type your current password.
5. In the **New password** field, type your new password.
6. In the **Confirm new password** field, retype your new password. Click **OK**.

Important:
The server administrator can disable the feature for changing your password from Designer. If the feature is disabled and you try to change your password, you will receive a message stating that the administrator has disabled the feature.

Synchronizing Passwords

The password stored locally in secure storage and in Integration Server can become out of sync. This mismatch of credentials occurs due to either of the following reasons:

- The password change operation is not successful due to machine or network failure between the time the new password is stored locally in secure storage and its new value is updated in Integration Server.
- The password to connect to Integration Server is changed using the change password functionality in another instance of Designer.

In both these instances, Designer will not be able to connect to Integration Server. To synchronize the passwords and reconnect to Integration Server, edit the disabled server definitions in Designer and provide the right credentials. For information on editing the server definitions, refer to "Editing Server Definitions" on page 31.

3 Working with Elements

- About Element Names 41
- Creating New Elements 41
- Guidelines for Working with Elements 43
- Opening Elements 43
- Closing Elements 44
- Editing and Saving Elements 44
- Adding Comments for an Element 45
- Configuring Dependency Checking for Elements 45
- Controlling the Reuse of Elements Published to CentraSite 46
- Allowing Editing of Derived Elements 47
- Moving and Copying Elements 48
- Renaming Elements 52
- Refactoring Variable Names in Elements 54
- Deleting Elements 61
- Finding Dependents and References 62
- Inspecting Pipeline References 65
- Finding Elements and Variables 67
- Filtering Displayed Elements 71
- Hiding or Displaying Automatically Generated Flow Services 71

- Creating Working Sets 72
- Caching Elements 72
- Exporting Elements 74
- Viewing Server Files for an Element 74
- Using Property Templates with Elements 74

An *element* is an item that exists in the Package Navigator view in Software AG Designer. Elements include folders, services, specifications, document types, triggers, and web service descriptors. In the Package Navigator view, servers and packages are not considered to be elements.

About Element Names

The fully qualified name of an Integration Server element is composed of two parts: a *folder identifier*, consisting of the folder path in which the element resides, and the *element name*. Integration Server represents elements in the following format:

folder.subfolder1.subfolder2:element

For example, if the HomeLoan service is in the Personal folder, which is contained in the Finance folder, the fully qualified service name is:

Finance.Personal:HomeLoan

Designer ensures that the fully qualified name of each element within the server is unique. Depending on the action you are performing on the element, Designer accomplishes this either by alerting you that the action cannot be completed or by appending a number to the name of the element after the action is performed. For example, if you are copying a flow service named checkOrder2 to a destination that already contains a flow service with that name, Designer copies the service and names the copy checkOrder2_1.

Package Names and Element Names

The name of the package to which an element belongs has no bearing on the names of the elements that package contains (that is, the package name is not part of the fully qualified name of the element). Nor does it affect how the element is referenced by a client application. For example, if you move a service called Personnel:GetDeptNames from a package called Admin to a package called EmployeeData, client applications would still reference the service as Personnel:GetDeptNames.

Creating New Elements

When creating elements, keep the following points in mind:

- The names of non-folder elements must be unique across all packages. If you try to create an element using a name that already exists at that level in any package, Designer creates the element and names it Untitled.
- Designer places some restrictions on the characters you can use in element and package names. For more information about these restrictions, see "Guidelines for Naming Elements" on page 42.
- Before you can create a new Java or C service, ensure that all services in the folder in which you want to create the new service and of the type you want to create (Java or C) are unlocked. Alternatively, you can ensure that you have all the services locked. For more information, see "Guidelines for Locking Java and C/C++ Services" on page 90.

To create a new element

1. In the Package Navigator view of Designer, click **File > New** and then click the element you want to create.

   You can also click **Other** to view all the elements that you can create in Designer. The **New** dialog box appears. Click the type of element you want to create and then click **Next**.

2. On the **New *Element*** dialog box, follow the prompts given by Designer for the type of element you are creating.

3. To create an element such as a flow service or a Java service using a predefined template, select the template from the **Choose template** list. If you want to apply the default properties to the element, select **Default** from the **Choose template** list.

   Note:
   The **Choose template** field appears only for those elements that support property templates. The default value for this field is Default.

4. When you have supplied all the information that Designer needs to create the element, click **Finish**. Designer refreshes the Package Navigator view and displays the new element.

Note:
When Designer creates web service connectors as part of creating a consumer web service descriptor, Designer applies the default property template to the web service connector. You can modify the properties of the element by changing them in the Properties view or apply a different template after the element is created. For more information about applying templates, see "Applying Property Templates to Elements" on page 76.

Guidelines for Naming Elements

Designer places some restrictions on the characters you can use in element and package names. Specifically, element and package names cannot contain:

- Reserved words and characters that are used in Java or C/C++ (such as *for*, *while*, and *if*).
- Digits as their first character (This does not apply to packages.)
- Spaces.
- Control characters and special characters like periods (.), including:

?	'	-	#	=	)	(	.	/	\	;
&	@	^	!	|	}	{	`	>	<	
%	*	:	$	]	[	"	+	,	~	

- Characters outside of the basic ASCII character set, such as multi-byte characters.

If you specify a name that disregards these restrictions, Designer displays an error message. When this happens, use a different name or try adding a letter or number to the name to make it valid.

Guidelines for Working with Elements

When performing actions on one or more elements, keep the following points in mind:

- You must have at least List access to view elements, Read access to select elements to move or copy, Write access to the location to which you want to move/copy elements, and Write access to elements you want to rename or delete. If you select multiple elements and you do not have the required access to one or more of them, you will not be able to perform the action. You must either ask your system administrator to give you the required access to the elements or select only elements for which you have the proper access.
- Designer prompts you to save changes to an element before allowing you to perform an action on the element, close the element in the editor, close your session on the current server, or exit Designer.
- The actions you can perform on items depend on the type and combination of items you select. If an action is not allowed for one or more elements in a selection, Designer makes the action unavailable for use. For example, Designer disables the cut, copy, paste, and delete actions if you select a server. Designer also prevents you from selecting multiple elements when doing so could cause confusion or undefined results. For example, you cannot select a server and any other element, a package and any other element, or a folder and one or more elements contained within that folder.
- If you select multiple elements and Designer encounters an error while performing the specified action on one or more of the elements, Designer displays a dialog box listing the elements for which the action failed. You can obtain more information about why the action failed by clicking **Details**.
- The elements you want to move, copy, rename, save, or delete must be unlocked, or locked by you. If you configured Integration Server to work with a version control system (VCS), you must first check out the element before editing it and then check it back in.
- You cannot undo a move, copy, rename, or delete action using the **Edit > Undo** command.
- If you select a publishable document type that is associated with an adapter notification, Designer handles actions performed on the document type as follows:
  - For *non-copy* actions, you must also select the adapter notification before you can perform a non-copy action on the document type.
  - For *copy* actions, you can select the publishable document type without selecting its associated adapter notification. However, the copied publishable document type loses its association with the adapter notification.

Opening Elements

When opening elements from the Package Navigator view, keep the following points in mind:

- Double-click a folder to expand or collapse the contents of the folder in the Package Navigator view.
- If you have enabled the Version Control System (VCS) Integration feature of Designer, Designer might exhibit slowdowns, error messages (such as “Server version has changed” and “Session already in use”), and may stop responding completely when you expand a large element (such as a folder) in the Package Navigator view. This condition occurs because Designer checks the lock status of each element within the expanded element in the Integration Server.

To open elements in the editor

- In Package Navigator view, double-click the element you want to open.

Tip:
You can also use the Open Integration Server Element dialog box to easily locate and open an element by typing any portion of the element name. To open the Open Integration Server Element dialog box, right-click anywhere in the Package Navigator view and select **Open Elements** from the context menu or press the CTRL+SHIFT+A keys in the keyboard.

Closing Elements

Keep the following points in mind when closing elements:

- You do not need to close elements when you exit Designer. Designer remembers which elements were open and displays them when you restart Designer.
- If you close an element without saving changes made to the element, Designer will prompt you to save changes.

To close an element

- Do one of the following:

To...	Click...
Close the active element (that is, the element whose tab is highlighted)	**File > Close**.
Close *all* elements	**File > Close All**.

Editing and Saving Elements

To edit an element, you must first lock it (or check it out). You must also have Write access to the element.

Changes that you make to an element are not written to Integration Server until you explicitly save your work. If you attempt to close Designer, close your session on the current server, close an unsaved element in the editor, or perform an action on an element without saving your changes, Designer will prompt you to save changes first.

To save changes to elements

- To save changes to elements

To...	Click...
Save changes to the current element	**File > Save**.
Save all elements you have edited, on all servers	**File > Save All**.

Adding Comments for an Element

If you want to include any instructions on usage of an element, descriptive comments, or notes, you can use the **Comments** tab.

Note:
The contents of the **Comments** property that was available in the Properties view in previous versions of Service Development are available in the **Comments** tab starting from version 9.7.

To add comments for an element

1. In the Package Navigator view, open the element for which you want to specify a comment.
2. Click the **Comments** tab.
3. Specify notes or comments about the element.
4. Click **File > Save**.

Configuring Dependency Checking for Elements

Designer automatically checks for dependents when you delete, rename, or move elements in the Package Navigator view. This dependency checking acts as a safeguard to prevent you from inadvertently affecting other elements on the Integration Server. This is especially important during collaborative development on the same Integration Server.

You can have Designer prompt you before deleting, moving, or renaming an element with dependents. You can also have Designer update local references when pasting elements.

The dependency checking options are enabled by default.

To specify dependency checking safeguards

1. In Designer, click **Window > Preferences**.

2. In the Preference navigation tree, click **Software AG > Service Development > Package Navigator**.

3. Under Preferences for the Package Navigator view, do the following:

Select...	To...
Confirm before deleting	Instruct Designer to notify you before deleting an element used by other elements, such as flow services, IS document types, specifications, or triggers. If Designer finds elements that depend on the element being deleted, Designer lists those dependents and prompts you to delete the element anyway or cancel the action. If you clear this check box, Designer deletes the element without prompting you.
Prompt before updating dependents when renaming/moving	Instruct Designer to alert you when dependents (that is, other elements that use the selected element, such as flow services, IS document types, or triggers) exist. If dependents exist, Designer lists those dependents before renaming or moving the selected element and prompts you to: ■ Rename/move the selected element and update references in dependent elements. ■ Rename/move the selected element without updating references to it. ■ Cancel the action. If you clear this check box, Designer automatically updates dependents without prompting you.
Update local references when pasting multiple elements	Instruct Designer to update references when copying and pasting a group of elements that refer to each other. If you clear this check box, Designer retains the original references in the copied elements.

4. Click **OK**.

Controlling the Reuse of Elements Published to CentraSite

You can control whether or not assets you have published to CentraSite can be reused in BPM processes or CAF projects. You do this by designating an asset as public or private.

In Designer, when an element is public, you can drag it from the CentraSite Registry Explorer view to a BPM process or CAF project. When an element is private, you cannot drag it from the CentraSite Registry Explorer view to a BPM process or CAF project, or any other location.

Use the public setting for assets you want to be reused more widely within your organization, beyond the bounds of the current project.

All published assets are available for Impact Analysis, whether they are private or public.

Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.

To control the reuse of the published asset for an element

1. In the Package Navigator view, select the element you want to work with.
2. In the Properties view, next to the **Reuse** property, do one of the following:

To...	Select...
Allow the published asset to be dragged from the CentraSite Registry Explorer view to a BPM process or a CAF project	**Public**
Prevent the published asset from being dragged from theCentraSite Registry Explorer view to a BPM process or a CAF project	**Private**

3. Click **File > Save**.

Allowing Editing of Derived Elements

For an element created from an external source, such as a document type created from an XML schema definition, you can control whether the content and structure of the element can be modified. When Integration Server creates the element, the **Linked to source** property is set to true which indicates that the element content accurately reflects the source file. When the **Linked to source** property is set to true, you can edit the element properties, however, you cannot edit the element content. For example, for a document type created from an XML schema definition, you can change the element permissions or storage type, but you cannot add or edit fields or change the order of fields in the document type.

If you want to edit the content of an element linked to its source, you must first break the association with source by setting **Linked to source** to false. A value of false indicates that the content and structure of the element may not reflect the source from which it was created. The content and structure can be edited.

Keep the following points in mind when allowing editing for elements derived from a source:

- Changing the value of the **Linked to source** property to false cannot be undone.
- You cannot change the **Linked to source** property from false to true.

To allow editing of a derived element

1. In the Package Navigator view, select the element for which you want to allow editing.
2. In the Properties view, next to **Linked to source**, select **False**.
3. When Designer prompts you to confirm the change, click **OK** if you want to proceed. Otherwise, click **Cancel**.
4. Click **File > Save**.

Moving and Copying Elements

You can move or copy elements between packages and, in most cases, across servers. Before moving or copying elements, review the following guidelines and notes:

- "Guidelines for Moving and Copying All Types of Elements" on page 50
- "Guidelines for Moving and Copying Services" on page 50
- "Guidelines for Copying Elements Between Servers" on page 51
- "Notes for Moving and Copying Adapter Notifications and Related Elements" on page 52
- "ACLs and Creating, Viewing, and Deleting Elements" on page 85

To move or copy elements

1. In the Package Navigator view, select the elements that you want to move or copy.
2. Do one of the following:

To...	Click...
Move the element	**Edit > Cut**.
Copy the element	**Edit > Copy**.

Tip:
You can cancel a cut action by pressing ESC.

3. If the elements you want to move or copy contain unsaved changes, Designer alerts you that you must first save the changes. Click **OK** to close the alert dialog box. Then, save the changes and repeat the move/copy action.
4. If you do not have Read access to the elements you are moving or copying, or Write access to the location you are moving/copying them to, Designer displays a message that identifies the

elements that are preventing the action from completing successfully. Click **OK** and then either obtain the proper access from your system administrator or select only those elements to which you have proper access.

5. Select the location where you want to move or copy the elements.

6. Click **Edit > Paste**.

7. If the destination already contains an element with the same name as an element you are moving or copying, do one of the following:

   - If you are *moving* the element, Designer alerts you that the element cannot be moved. Click **OK** to close the alert dialog box. Rename the element if desired and repeat the move action.
   - If you are copying the element, Designer copies the element and appends a number to the name of the copied element. (For example, if you are copying a flow service named checkOrder2 to a destination that already contains a flow service with that name, Designer copies the service and names the copy checkOrder2_1.) Rename the element if desired.

   For more information about renaming elements, see "Renaming Elements" on page 52.

8. If one of the elements you moved or copied is a Java service, perform the following as necessary:

   - If you are moving or copying the Java service to a folder with other Java services that are system locked or locked by another user, Designer alerts you that the element cannot be moved/copied. Click **OK** and then ask the owner of the lock to remove the lock.
   - If the Java service you are moving or copying contains a shared source that conflicts with the shared source of an existing Java service in the destination folder, Designer alerts you that there is a conflict. Click **OK** to use the destination folder's shared source, or click **Cancel** to cancel the entire move action.

     Note:
     If no shared Java source conflict exists, Designer moves the Java service and its shared source to the destination folder. If a conflict does exist, you must re-specify the shared source in the copy of the service. Using the Designer Java Service Editor, you can copy the information from the **Source** tab of the original service to the **Source** tab of the copy. For more information about the **Source** tab of the Java Service Editor, see "Source Tab" on page 305

9. If you selected the **Prompt before updating dependents when renaming/moving** check box in the Package Navigator preferences and any dependent elements on the current server contain unsaved changes, Designer alerts you to save them. Select the elements and click **OK** to save the changes or **Cancel** to cancel the entire move or copy action.

10. If the **Move and Rename Dependencies** dialog box appears, do one of the following:

To...	Click...
Move the selected element and update references to dependent elements	**Update Usages**
Move the selected element in the Package Navigator view without updating references to dependent elements	**Ignore Usages**
Cancel the entire move action	**Cancel**

For more information about dependency safeguards, see "Configuring Dependency Checking for Elements" on page 45.

Tip:
You can also move elements by clicking and dragging them to their new location.

Guidelines for Moving and Copying All Types of Elements

- You must have Read access to the elements you are moving or copying and Write access to the packages, folders, or servers to where you want to move/copy them. For more information about Write access and ACLs assigned to elements, see "Assigning and Managing Permissions for Elements" on page 79.
- When you move or copy an element, Designer automatically changes the element's fully qualified name to reflect its new location.
- You cannot *move* an element to a location that already contains an element with the same name. If you *copy* the element, however, Designer copies the element and appends a number to the end of the name of the copied element.
- You cannot move multiple elements with the same name to a single location.
- When you *copy* multiple elements to another location on the same server and the elements contain references to each other, Designer updates the references if you have selected **Update local references when pasting multiple elements** preference. For example, if you copy a folder that contains two services and one of the services invokes the other, Designer will update the reference to the invoked service.

Guidelines for Moving and Copying Services

- When you move or copy a service, Designer does not move/copy any output templates that are associated with that service.
- If you *move* a service, or a folder containing a service, Designer retains the service's explicit universal name. If you *copy* a service or a folder containing a service, Designer does not retain the service's explicit universal name. You must restore the universal name by editing the service's properties. For more information, see "About Universal Names and Document Types" on page 552.

- When you move or copy a Java service, Designer automatically recompiles the service and any Java services that remain in the source folder. When you delete a Java service, Designer recompiles any Java services that remain in the source folder.
- You cannot move or copy a Java service to a folder that contains other Java services that are system locked or locked by another user. If you attempt to do so, Designer cancels the entire move/copy action.
- When you move or copy a Java service, Designer will also move or copy the service's Shared fields to the destination folder, unless the destination folder already contains Shared fields with different values. In this case, you must first manually copy the Shared fields into the destination folder and then move or copy the Java service

Guidelines for Copying Elements Between Servers

- You cannot copy or move a web service descriptor element between servers.
- When you cut and paste or drag elements between servers, Designer retains a copy of the elements on the source server. That is, a move (cut and paste or drag) action is the same as a copy action.
- Designer does not automatically copy an element's references to the destination server. Instead, it displays a dialog box after the copy alerting you to any unresolved references. You must copy the references to the destination server manually.
- Designer does not automatically update references when copying across servers. Therefore, if you are copying multiple elements from one server to another using Designer and the elements reference each other, you should paste the elements into a location with the same name on the destination server.
- If you are copying an add-in element that has a component that resides on the server, and the destination server does not have that add-in component installed, Designer displays an error message stating that you are attempting to copy an unknown element. Designer does not copy the add-in elements but does copy other elements in the selection. Elements you copy to a folder on a different server adopt the ACL access permissions of the destination folder, even if they had explicitly assigned ACLs on the source server. Folders you copy to a package on a different server inherit the default ACLs for top-level folders.
- When you copy a webMethods Messaging Trigger to another server, the trigger will be pasted in a disabled state. To create the subscriptions identified in the trigger, you must enable the trigger. When you copy a package to another server, the triggers contained in the package will maintain their original state.
- If you are configuring a cluster, use the package replication feature in the Integration Server Administrator to populate the cluster nodes. See *webMethods Integration Server Administrator's Guide* for more information about this feature.
- When you move or copy a publishable document type to a destination on the *same* server, the moved or copied document type remains publishable. When you copy a publishable document type to a *different* server, Designer converts the publishable document type to an IS document type on the destination server. For more information about making IS document types

publishable and synchronizing them with Broker document types, see "Working with Publishable Document Types" on page 553.

Tip:
To retain the status of a publishable document type and its link to a Broker document type, use the package replication functionality in the Integration Server Administrator instead of using Designer to move or copy the package containing the publishable document type. For information about package replication, see *webMethods Integration Server Administrator's Guide*.

Notes for Moving and Copying Adapter Notifications and Related Elements

- Although you cannot *move* an adapter notification's publishable document type without also moving its associated adapter notification, you can *copy* it. If you do so, the copied document type remains publishable but is no longer associated with the adapter notification.
- When you move or copy an adapter notification, Designer also moves/copies its associated publishable document type and prompts you to indicate whether to move/copy the associated Broker document type.
- You cannot move or copy adapter notifications, adapter notification publishable document types, or adapter services across servers. If you are selecting multiple elements and your selection contains any of these elements, Designer alerts you that the move/copy action cannot be completed.
- You cannot move or copy a listener or connection element.

Renaming Elements

When renaming elements, keep the following points in mind:

- You can rename any elements for which you have Write access to the element and its parent folder. When renaming a folder, you must also have Write access to all elements within the folder. For more information about Write access and ACLs assigned to elements, see "Assigning and Managing Permissions for Elements" on page 79.
- When you rename a folder, Designer automatically renames all of the elements in that folder (that is, changes their fully qualified names).
- If the folder you want to rename contains elements with unsaved changes, you must save the changes before you can rename the folder. You cannot rename a folder if elements within the folder are locked by other users.
- Element names must be unique across all packages. If you try to rename an element using a name that already exists, Designer reverts the element back to its original name.
- When you rename an adapter notification, Designer also renames its associated publishable document type and prompts you to indicate whether to rename the associated Broker document type.

- You cannot rename a listener or connection element.
- When you rename a publishable document type, Designer checks for dependents such as triggers and services that use the publishable document type. (Designer performs dependency checking only if you select the **Prompt before updating dependents when renaming/moving** preference.) If Designer finds elements that use the publishable document type, Designer gives you the option of updating the publishable document type name in each of these elements. If you do not update the references, all of the references to the publishable document type will be broken.

  Important:
  You must manually update any services that invoke the pub.publish services and specify this publishable document type in the *documentTypeName* or the *receivedDocumentTypeName* parameter.

Important:
If you need to change the name of a field in an element, refactor the field instead of renaming it. Refactoring ensures that usages of the field are updated with the new name, including uses in the pipeline such as mapping. For more information about refactoring, see "Refactoring Variable Names in Elements" on page 54

To rename an element

1. In Package Navigator view, select the element that you want to rename. Right-click the element and click **Rename**.

2. If the element you want to rename contains unsaved changes, Designer alerts you that the element cannot be renamed until you save the changes. Click **OK** to close the alert dialog box. Then, save the changes and repeat the rename action.

3. Edit the name and press ENTER.

   If an element already exists with that name at the same level, Designer displays a message alerting you that the rename action could not be completed. Click **OK** to close the message dialog box and repeat the rename action.

4. If you selected the **Prompt before updating dependents when renaming/moving** check box in the Package Navigator preferences and any dependent elements on the current server contain unsaved changes, Designer alerts you to save the elements that will be affected by the rename action. Select the elements and click **OK** to save the changes or **Cancel** to cancel the entire rename action.

5. If the **Move and Rename Dependencies** dialog box appears, do one of the following:

To...	Click...
Rename the selected element in Package Navigator view and update references to dependent elements	**Update Usages**.

To...	Click...
Rename the selected element without updating references to dependent elements	**Ignore Usages**.
Cancel the entire rename action	**Cancel**.

For more information about dependency safeguards, see "Configuring Dependency Checking for Elements" on page 45.

Refactoring Variable Names in Elements

If you want to rename a variable in an element, you may need to refactor the variable instead of simply renaming. When you refactor a variable, Designer updates all the usages of that reference in flow services, including pipeline mapping, IS document types, or other elements.

Renaming without refactoring can result in broken or invalid references, which are sometimes called "ghost references". The reference is broken or invalid because in places where the variable is subsequently used, such as mapping variables or assigning a value to a variable in the pipeline, Designer does not update the internal path that it maintains to the variable with the new variable name. Common situations where a broken variable reference occur are:

- A variable in a service signature is renamed and that variable was already mapped to another variable in the pipeline for an INVOKE or MAP step. With only a rename, the variable may exist twice in the pipeline (once with the old name and once with the new name). Additionally, it's possible that only the variable with the old name will retain the mapping.
- A flow service uses a document reference in the input signature. In an INVOKE step within the flow, a variable in the document reference is mapped to a service input. A variable in the document type to which the document reference points is renamed. The renamed variable is the same variable that is mapped in the flow service. Because the variable in the document type was merely renamed, the INVOKE pipeline shows a variable with the old name and a variable with the new name. The variable with the old name retains the mapping to the invoked service. If the document type variable had been refactored, only the variable with the new name would appear in the pipeline and the mapping would be updated.
- A value is assigned to a variable in the pipeline. The variable, which may be defined in a service signature, a document type reference, specification, etc., is renamed, but not refactored, in the original location. The pipeline view shows variables with the original name and the new name. The variable with the original name retains the assigned value. If the variable had been refactored, only the variable with the new name would exist and it would have the assigned value.
- You add a variable to the pipeline, assign a value to it by linking it to another variable or using to set a value, and then rename the variable. The pipeline will have the variable with the new name and the old name. The variable with the old name retains the mapping. Because the pipeline view does not allow refactoring, you must manually update any usages.
- A variable is renamed and that variable is used in flow step properties, mapping, or in local filters for a webMethods messaging trigger. Merely renaming the variable does not update

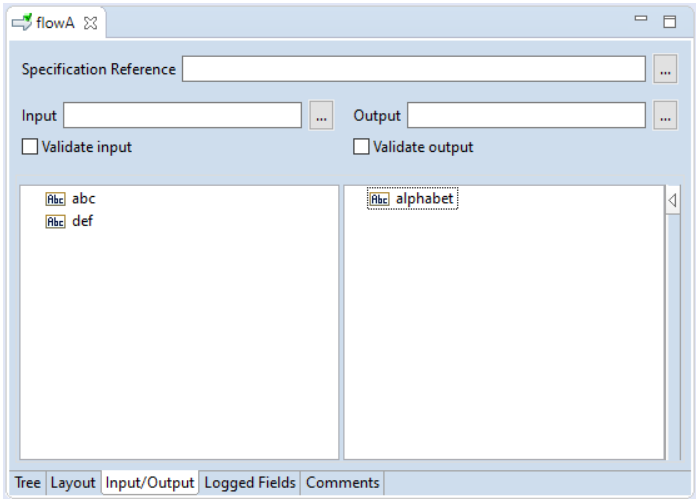
the locations where the variable was used as part of variable substitution. When the variable is refactored, Designer locates and updates variable substitution usages. Note that Designer does not support refactoring of variable in the pipeline.

- Changing the variable data type after the variable is used in pipeline mapping, variable substitution, or in local filters for a webMethods messaging trigger.

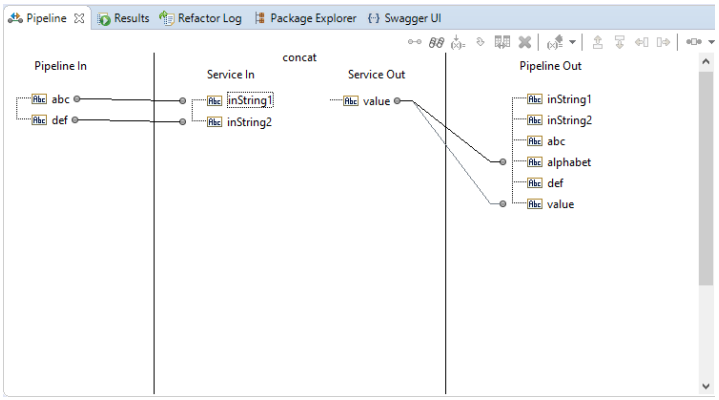
Example of a Broken Reference

Following is a simplified example of how renaming a variable can introduce a broken reference.

1. The flow service input signature contains two String variables named *abc* and *def*. The output signature contains a String variable named *alphabet*.



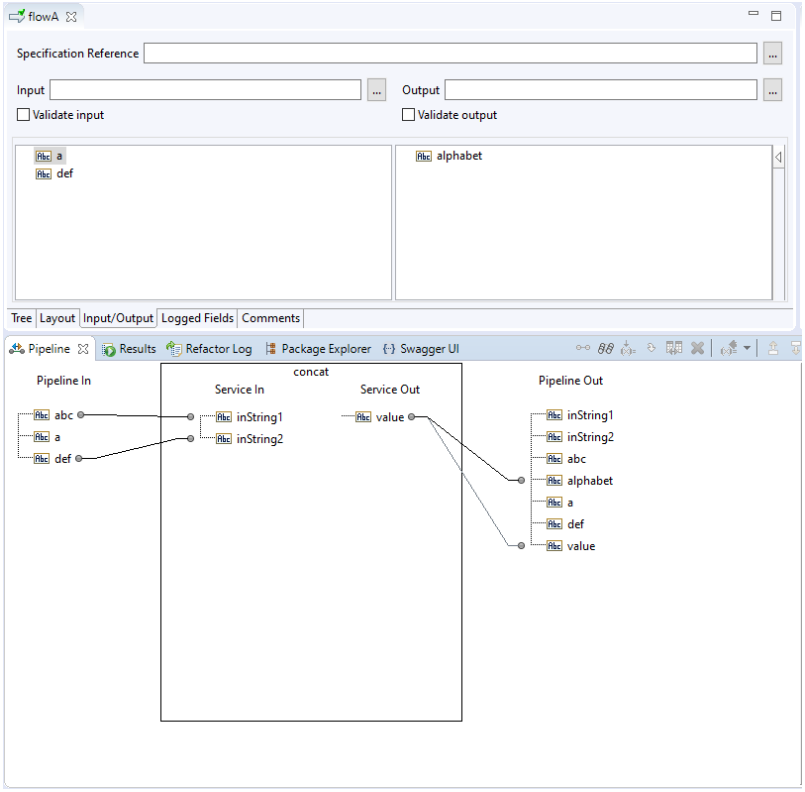
2. The flow service invokes the pub.string:concat. The String variables *abc* and *def* are mapped to *inString1* and *inString2*, respectively.



3. In the **Input/Output** tab of the flow service, the input variable *abc* is renamed to *a*. Rename is accomplished by right-clicking the variable and entering a new name.

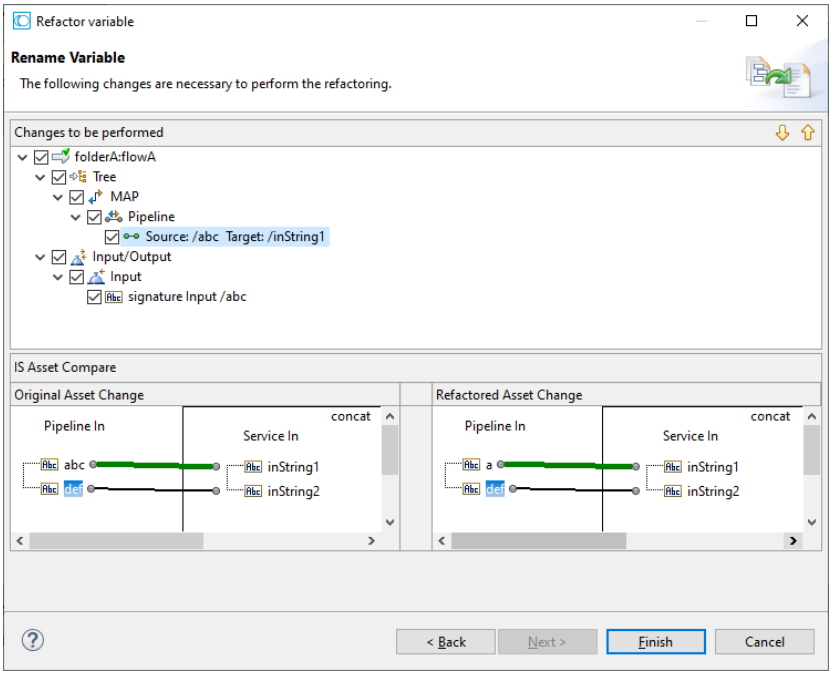
   In the pipeline for the invocation of pub.string:concat, notice that under **Pipeline In**, there are three variables: *abc* (old variable name), *a* (new variable name), and *def* (unchanged input).

The variable *abc* retains the mapping to the pub.string:concat input *inString1*. Variable *abc* now is a broken reference and the mapping is invalid.

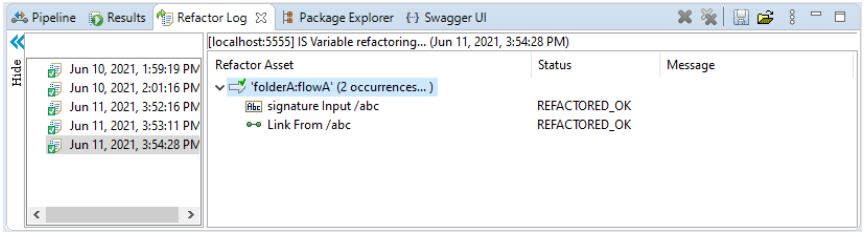


In comparison, if the *abc* variable had been refactored, a broken reference would not have been introduced.

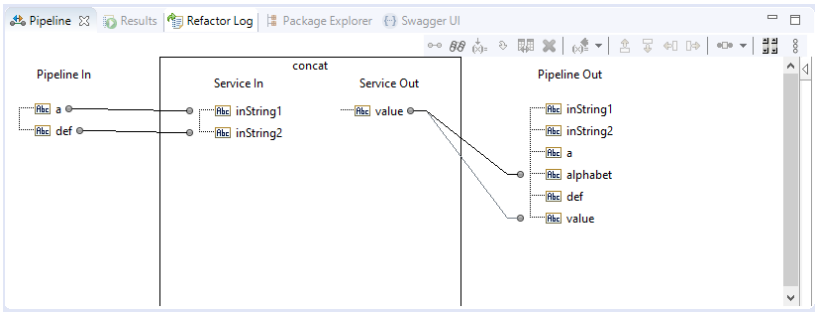
1. Refactor the variable by right clicking the variable and selecting **Refactor > Rename**, Designer prompts you to rename the variable and then displays all the usages that need to be refactored to reflect the new name.



2. After you click **Finish**, Designer refactors the usage of variable *abc*, displaying a log when it is done.



3. When viewing the pipeline for the INVOKE step, you can see that the renamed variable *a* exists and is mapped to *inString1* as *abc* used to be.



Guidelines for Refactoring Variables

You can refactor variable (field) names within Integration Server elements such as IS document types, flow services, Java services, specifications, and triggers. Refactoring variable names involves

renaming the variable and then updating usages of the variable that occur elsewhere, such as in the pipeline, other flow services, or document types.

The following table identifies the IS elements in which variable names can be refactored by Designer automatically.

Element	What can be refactored?
IS Document Types	Variable names in a document type.

Flow services	Variable names in the signature.
	
	Variable names used for variable substitution in the following properties of flow steps:
	
	■ **Label** property in all flow steps
	■ **Scope** property in all flow steps
	■ **Switch** property in the BRANCH step
	■ **Input array** and **Output array** properties in a LOOP step
	■ In a MAP step:
	  ■ Indices used in a link when mapping arrays
	  ■ Copy condition properties of linked variables
	  ■ Map set (assigned values and links) and map delete (dropped variables)
	■ **Timeout** property of the flow steps, BRANCH, CATCH, FINALLY, INVOKE, LOOP, MAP, REPEAT, SEQUENCE, and TRY
	■ **Count** and **Repeat interval** properties of REPEAT step
	■ **Failure message**, **Failure name**, **Failure instance** properties of the EXIT step

Java services	Variable names in the signature. **Note:** Refactor within Java code is not supported.
Specifications	Variable names in the signature.
Triggers	Variable names used in a local filter for a webMethods messaging trigger. The variable names used in a filter must be enclosed within %. If variable names are not

Element	What can be refactored?
	enclosed in %, you must manually change the variable name. **Note:** Provider filters cannot be refactored. You must manually change provider filters.

In addition to the above list of IS elements in which a variable can be refactored, keep the following guidelines and limitations in mind:

- To refactor variable names in an IS element, you must have write access to the element and its parent folder and have the element locked for edit (if locking is in use).
- You must refactor a variable in the source element and not in dependent elements or locations.
- You cannot perform a rollback of the changes after you refactor a variable. Software AG recommends that you back up your packages before refactoring.
- You cannot refactor local variables (variables added in the Pipeline tab using).
- You cannot refactor variable names in the pipeline. Designer allows rename only in the pipeline. However, if you rename the variable here, you may create a broken reference.
- If you refactor a variable in a publishable document type, you must resynchronize the publishable document type with its provider definition.
- Any variable occurrence that cannot be refactored by Designer must be refactored manually. That is, you need to find usages of the variable, update the variable name, and then change or redo the mapping of that occurrence (for a variable usage in the pipeline). Note that even after a manual update of a usage, the link that Designer maintained to the original variable name is lost forever. Any future rename of the original variable require manual updates to the same occurrences that could not be refactored manually.
- When you refactor variables, Designer provides a preview of all the elements selected for refactoring except messaging triggers. Designer refactors the usages even if the element is locked by another or the user doing the refactoring does not have Write access to the elements.
- For flow services and IS document types, Designer displays the **IS Asset Compare** view which shows the changes between the original asset and the refactored asset.
- When you refactor a variable that shares the same name and type with another variable, the preview lists both the variables. Carefully inspect the flow service to ensure that you select the right variable for refactoring.

Refactoring Variables in Elements

Refactoring a variable in an element involves renaming a variable and propagating that change to all the elements that use the source element. Refactoring ensures that changes are applied to all the applicable references of elements.

Before refactoring any variable in a element, make sure to review the considerations and limitations presented in "Guidelines for Refactoring Variables" on page 57.

Important:
You cannot perform a rollback of the changes after you refactor a variable. Software AG recommends that you back up your packages before refactoring a variable name.

To refactor variables in elements

1. In Package Navigator view, double-click the element to open it.

   Ensure that you have write access to the element.

2. Right-click the variable you want to refactor.

3. Select **Refactor > Rename**.

4. In the Refactor variable wizard, type the new name in the text box and click **Next**.

   All the occurrences of the variable are displayed in the **Changes to be performed** list. Designer provides a preview all the element types selected for refactoring *except* webMethods messaging triggers.

5. Select the check boxes next to the variable occurrences that you want to refactor. Designer selects all the variable occurrences by default.

   Note:
   If you clear the selection of a variable occurrence from the **Changes to be performed** list, that occurrence is unlinked from the variable in the source element. Then you must manually edit the particular variable occurrence. Note that the link to the variable in the source element will be lost forever and all future update to the variable occurrence need to be done manually.

   The **IS Asset Compare** view displays the changes between the original asset and the refactored asset. This view is only available for IS document types and flow services.

6. Click **Finish**. The **Refactor Log** tab displays the refactor status of each occurrence.

   - For variable occurrences that are not refactored, the refactor log displays the reason along with the status. You must manually update these occurrences. However, elements that are not refactored by Designer are unlinked from the source element. The unlinking cannot be undone which is why Software AG recommends that you back up your packages before refactoring a variable name.
   - For an element that has more than one input variable with the same name, refactoring any one of those variables results in Designer displaying an extra variable. This is an issue with the way that Designer displays variables with the same name after refactoring. The issue does *not* impact the results of the refactoring operation.

Deleting Elements

When deleting elements, keep the following points in mind:

- You can delete any elements to which you have Write access for the element and its parent folder. When deleting a folder, you must also have Write access to all elements within the folder. For more information about Write access and ACLs assigned to elements, see "Assigning and Managing Permissions for Elements" on page 79
- When you delete a folder or the last Java service in a folder, Designer also deletes the shared source for that folder. If you cancel the delete action, no elements (including non-Java service elements) are deleted.
- You can only delete an adapter notification's publishable document type if you delete its associated adapter notification.
- When you delete an adapter notification, Designer also deletes its associated publishable document type and prompts you to indicate whether to delete the associated Broker document type.
- You cannot delete a listener or connection element.
- If you delete a dictionary, the dependency manager will list all flat file schemas and dictionaries that will be impacted by the deletion, and prompts you to confirm the deletion. However, it does not identify the names of the records, fields, or composites that reference the dictionary; that is your responsibility.
- If you delete a publishable document type, Designer prompts you to keep or delete the associated Broker document type.
  - If you delete a publishable document type and Broker document type associated with a trigger or a flow service, you might break any integration solution that uses the document type.
  - If you delete the Broker document type, you might negatively impact any publishable document types created from that Broker document type on other Integration Servers. When the developers synchronize document types with Broker and they choose to **Pull from Broker**, publishable document types associated with the deleted Broker document type will be removed from their Integration Servers.

To delete elements

1. In Package Navigator view, select the elements that you want to delete.
2. Select **Edit > Delete**.
3. If you have selected the **Confirm before deleting** check box in the **Preferences** dialog box for Package Navigator view, do the following:

a. If any elements on Integration Server have unsaved changes, Designer prompts you to save changes. Select the elements whose changes you want to save and click **OK**.

b. If other elements are dependents of the elements you are deleting, Designer indicates which items will be affected by the deletion.

c. If you are deleting a publishable document type, Designer prompts you to keep or delete the associated Broker document type. Do one of the following:

To...	Do this...
Delete the publishable document type on Integration Server but leave the corresponding document type on the Broker	Clear the **Delete associated Broker document type on the Broker** check box.
Delete the publishable document type on Integration Server *and* the corresponding document type on the Broker	Select the **Delete associated Broker document type on the Broker** check box.

d. Click **Continue** to delete the selected elements.

4. If you did not select the **Confirm before deleting** preference and one of the elements that you want to delete is a publishable document type, Designer prompts you to keep or delete the associated Broker document type. Select the **Delete associated Broker document type on the Broker** check box, if you want to delete the publishable document type and the Broker document type. Click **OK**.

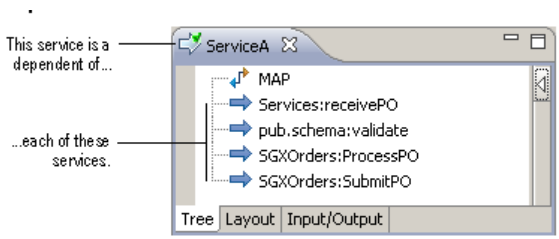
Finding Dependents and References

Before performing an action on a selected element, you can determine whether other elements will be affected by the change by finding dependents and references of the element. In Designer, a ***dependent*** is an element that *uses* a selected element, and a ***reference*** is an element that *is used by* a selected element.

What Is a Dependent?

To determine how a selected element is used by other elements on the server, you can find ***dependents*** of the selected element. A ***dependent*** is an element that *uses* a selected element. For example, suppose that the flow service ServiceA invokes the flow service receivePO. The ServiceA service *uses* the receivePO service. This makes ServiceA a ***dependent*** of the flow service receivePO. If you delete receivePO, ServiceA will not run.

Dependent elements



During debugging, you might want to locate all of the dependents of a given service or IS document type. Or, before editing an IS document type, you might want to know what elements, such as specifications, webMethods Messaging Triggers, or flow services, will be affected by changes to the IS document type.

In addition to finding a dependent IS element, Designer also finds the Dynamic Server Pages (DSPs) that depend on the service. For example, suppose that the DSP page myPage.dsp resides in *Integration Server_directory* \instances*instance_name*\packages\myPackage\pub and uses the service myFolder:submitMyPage. If you find dependents for the myFolder.submitMyPage service, Designer also returns the following as a dependent:

Integration Server_directory \instances*instance_name*\\packages\myPackage\pub\myPage.dsp

Note: Designer does not consider a Java service that invokes another services to be a dependent. For example, if Java service **A** invokes service **B**, and you instruct Designer to find dependents of service **B**, service **A** will not appear as a dependent.

Finding Dependents

To find dependents of a selected element

1. In Package Navigator view, right-click the element for which you want to find references and select **Find Dependents**.

2. If any elements on Integration Server have unsaved changes, Designer prompts you to save changes. Select the elements whose changes you want to save, and then click **OK**.

   Designer displays the dependents of the selected element on the Search view.

3. After Designer finds the dependents of the selected element, you may do either of the following:

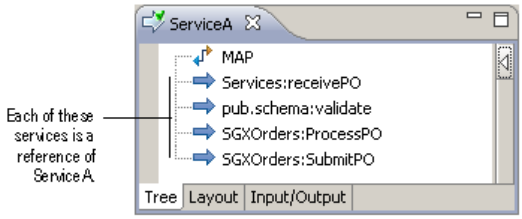
   - To jump to an element in Package Navigator view, right-click that element in the results, and select **Show In > Package Navigator**.
   - To see all dependents of a found dependent click ⊞ next to the item in the results list.

What Is a Reference?

To determine how a selected element uses other elements on the server, you can find *references* of the selected element. A *reference* is an element that *is used by* a selected element. For example,

the flow service ServiceA invokes the services receivePO, pub.schema:validate, processPO and submitPO. Additionally, in its input signature, ServiceA declares a document reference to the IS document type PODocument. The services receivePO, validate, ProcessPO, and SubmitPO, and the IS document type PODocument, *are used by* ServiceA. The elements receivePO, validate, ProcessPO, SubmitPO, and PODocument are *references* of ServiceA.

Elements as references



During debugging of a complex flow service, you might want to locate all of the services, IS document types, and specifications used by the flow service. Use the **Find References** command to locate the references.

You can also use the **Find References** command to locate any unresolved references. An *unresolved reference* is an element that does not exist in the Package Navigator view yet is still referred to in the service, IS document type, or specification that you selected. The element might have been renamed, moved, or deleted. To prevent unresolved references, specify the dependency checking safeguards. For more information about these safeguards, see "Configuring Dependency Checking for Elements" on page 45.

Note: Designer does not consider document references to schema types to be references, nor does it consider services invoked within a Java service to be references of the Java service. For example, if Java service **A** invokes service **B**, and you instruct Designer to find references for service **A**, service **B** will not appear as a reference of **A**.

Finding References

To find references of a selected element

1. In Package Navigator view, right-click the element for which you want to find references and select **Find References**.

2. If any elements on Integration Server have unsaved changes, Designer prompts you to save changes. Select the elements whose changes you want to save, and then click **OK**.

   Designer displays the references of the selected element on the Search view.

3. After Designer finds the references of the selected element, you may do either of the following:

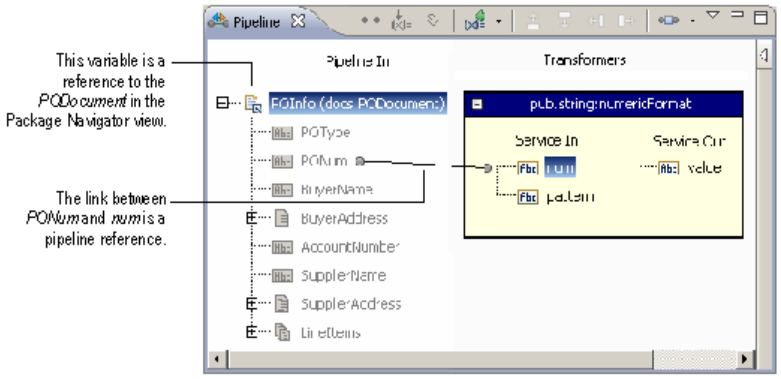
   - To jump to an element in Package Navigator view, right-click that element in the results, and select **Show In > Package Navigator**.

- To see all references of a found reference, click ⊞ next to the item in the results list.

Inspecting Pipeline References

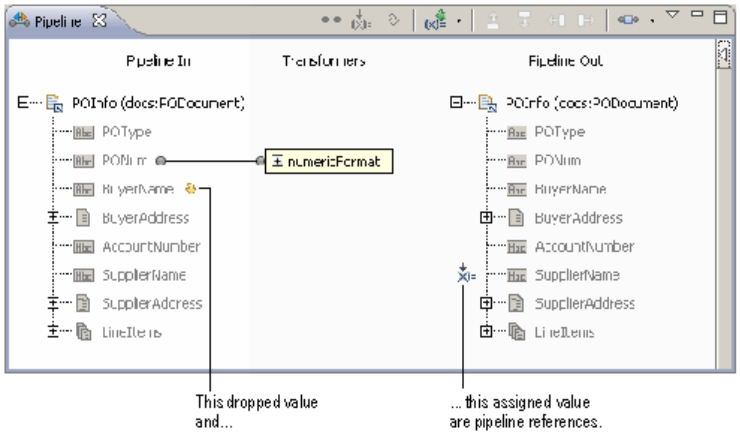
A *pipeline reference* is where a variable in a document reference or document reference list in Pipeline view is linked to another variable, assigned a value, or dropped. For example, in its input signature, ServiceA declares a document reference to the IS document type PODocument. If ServiceA contains an INVOKE or MAP step in which a variable in the document reference is linked to another pipeline variable, then that link is a pipeline reference. In the following illustration of the Pipeline view, the link between *PONum* and *num* is a pipeline reference.

Pipeline reference



Pipeline references are also those locations where you modify the value of a variable in a document reference or document reference list by assigning a value using (x)= or dropping a value using on the Pipeline view toolbar. The following image of Pipeline view identifies these types of pipeline references.

Examples of pipeline references



When you edit an IS document type, the changes affect any document reference and document reference list variables defined by that IS document type. The changes might make pipeline references invalid. For example, suppose the input signature for ServiceA contains a document reference variable *POInfo* based on the IS document type PODocument. The IS document type

PODocument contains the field *PONum*. In the pipeline for ServiceA, you link the *PONum* field to another pipeline variable. If you edit the PODocument IS document type by deleting the *PONum* field, the pipeline reference (the link) for the field in the ServiceA pipeline is broken (that is, it is invalid) because the pipeline contains a link to a field that does not exist.

When you edit an IS document type, you might want to check all dependent pipeline modifiers for validity. You can use the **Inspect Pipeline References** command to locate any broken or invalid pipeline references. You can use this command to:

- Search for invalid pipeline references in a selected flow service.
- Search for invalid pipeline references involving document reference and document reference list variables defined by a selected IS document type.

Inspecting Pipeline References

When inspecting pipeline references, keep the following points in mind:

- You can inspect pipeline references in a selected flow service. You can also inspect pipeline references for document reference or document reference list variables based on a selected IS document type. The search results include only flow services, document reference variables, or document reference list variables that contain invalid pipeline modifiers.
- Values assigned to document reference or document reference list variables in the pipeline are not considered pipeline references. That is, when a value is assigned to a document reference or document reference list variable using on the Pipeline view toolbar, Designer does not treat the assigned value as a pipeline reference.
- The search results will not show data type and dimensionality mismatches. For example, suppose that you link a String named *Number* to the *PONum* String list within the document reference *PODocument*. This dimensionality mismatch will not appear in the search results.
- When you inspect pipeline references in a flow service, Designer inspects references across all packages on Integration Server.
- When you inspect pipeline references for an IS document type, you can inspect references across a specific package or all packages.

To inspect pipeline references

- In Package Navigator view, right-click the flow service or IS document type for which you want to find invalid pipeline references and select Inspect Pipeline References.

  The Search view displays all invalid pipeline references for the selected service or IS document type.

  - If you inspected a flow service, the search results contain all of the document references that have invalid pipeline references in that flow.
  - If you inspected an IS document type, the search results contain all of the flow services that have invalid pipeline references to that IS document type.

After Designer finds the pipeline references of the selected element, you can jump to an element in Package Navigator view. Right-click that element in the results, and select **Show In > Package Navigator**.

Finding Elements and Variables

You can use Designer to find elements and variables within elements using the following methods:

- **Searching for elements in the Package Navigator view.** When creating and editing elements, you might lose track of where you saved certain elements. For example, suppose that you do not remember the folder to which you saved a service called Test. You can use either the Search dialog box or the Open Integration Server Element dialog box to search for elements.
- **Locate an invoked service from the editor.** You can highlight the location of an invoked service in the Package Navigator view. This is especially helpful when working with a flow written by another party or complex flows that make multiple invokes.
- **Locate a referenced document type from the editor.** You can highlight the location of a referenced document type in the Package Navigator view. This is especially helpful when working with services with complex signatures, mapping data, or working with flow services written by other developers.
- **Linking open editors.** If you have a lot of elements open, you might want to quickly bring the editor for an element to the top of the stack of open editor views.

Searching for Elements or Variables in Package Navigator

You can search Package Navigator for elements and for specific variables in elements. For guidelines regarding searching for variables, see "Guidelines for Searching for Variables in Elements" on page 68

Note:
You cannot search for Trading Networks (TN) document types on Integration Server.

To search for elements or variables in Package Navigator

1. In Designer: **Search > Integration Server**.
2. In **Search string**, type any portion of the fully qualified name of the element that you want to find. You can use valid Perl regular expressions to specify the search string.
3. If you want to further refine the search string, select the following options.
   - To perform a case-sensitive search of element names, select the **Case sensitive** check box.
   - To search for an element with the name that exactly matches the specified search string, select the **Exact match** check box.

4. In the **Search on this server only** list, select the Integration Server to which you want to limit the scope of the search.

5. Define the package to search by using any of the following options.

If you want to...	Then...
Limit the scope of the search to a specific package	Select the package in the **Search in this package only** list.
Search within all the user-defined packages	a. Select **All Packages** in the **Search in this package only** list. b. Clear the **Include system packages** check box.
Search within all the packages	a. Select **All Packages** in the **Search in this package only** list. b. Select the **Include system packages** check box.

6. Click **Browse** and select the asset types to search for the specified search string.

   The selected asset types are displayed as comma-separated values in the **Asset type** text box.

7. Click **Search**.

   The search results are displayed in the Search view.

Guidelines for Searching for Variables in Elements

You can search for variables (fields) within Integration Server elements and define the scope of the search operations. Search can be performed on IS document types, flow services, Java services, specifications, and triggers.

The following table identifies where you can search for field names in specific Integration Server elements.

Element	What can you search for?
IS Document Types	Variable names in a document type.
Flow services	Variable names in the service signature. Variable names used for pipeline variable substitution in the following properties of flow steps: ■ **Label** property in all the flow steps

Element	What can you search for?

	■ **Scope** property in all the flow steps
	■ **Switch** property in the BRANCH step
	■ **Input array** and **Output array** properties in a LOOP step
	■ In a MAP step
	  ■ Indices of link
	  ■ Copy condition properties of linked variables
	  ■ Map set (assign a value to a variable) and map delete
	■ Timeout property of the flow steps: BRANCH, CATCH, FINALLY, INVOKE, LOOP, MAP, REPEAT, SEQUENCE, and TRY
	■ **Count** and **Repeat interval** properties of REPEAT step
	■ **Failure message**, **Failure name**, **Failure instance** properties of the EXIT step
	Note:
	You can search only on the variable names used in any of the specified flow steps.

Java services	Variable names in the signature. **Note:** Search within Java code is not supported.
Specifications	Search for field names in the signature.
Triggers	Variable names used in a local filter of a webMethods messaging trigger. The variable names used in a filter can only be searched if they are enclosed within %. **Note:** Provider filters cannot be searched. You must manually examine provider filters.

Note:
If a document type is referenced or used in other elements, the field names inside the reference or usage are not searched. For example, suppose an IS document type named address includes a field named *country* and the address document type is referred to by a document reference named *billingAddress* and is used in the signature of a flow service named processOrder. Searching

for *country* returns only address. The result set does not include the *billingAddress* document reference or the processOrder flow service.

Performing a Quick Search of Elements in Package Navigator

You can quickly search for an element in an Integration Server that you are connected to using the Open Integration Server Element dialog box. You can search for an element in only one Integration Server at a time.

You can use asterisks (*) to search for any string and question mark (?) to search for any character in the Open Integration Server Element dialog box. You can also use appropriate uppercase or mixed case letters to specify search patterns. For example:

- Specify `PO` for element names such as ProcessOrders, which contain the letters "P" and "O".
- Specify `GePrO` for element names such as GetProcessOrders, which contain the letters Ge, Pr, and O.

To perform a quick search of elements

1. Right-click anywhere in the Package Navigator view and select **Open Elements** from the context menu. Designer displays the Open Integration Server Element dialog box.

2. In the **Select an element to open** field, type any portion of the name of the element that you want to open. Designer displays the list of elements in the Integration Server that correspond to the name or search pattern you type.

3. Select the element and click **Open** to open the element in the editor. To select multiple elements, press the CTRL key while selecting.

   Note:
   You can also select an element and click **Show in > Package Navigator** to locate the element in Package Navigator view.

Locating Invoked Services

To locate an invoked service

1. In the flow editor, select the INVOKE step containing the service you want to locate.

2. Select **Navigate > Show In > Package Navigator**. Designer locates and selects the service in Package Navigator view.

Locating Referenced Document Types

To locate a referenced document type

1. In the editor, select the document reference that you want to locate.
2. Select **Navigate > Show In > Package Navigator**. Designer locates and selects the IS document type in the Package Navigator view.

Linking Open Editors

To link open editors

1. On the Package Navigator toolbar, click .
2. In Package Navigator view, select the element whose editor you want to view. If the element is open, Designer brings its editor to the front. Designer displays the element in the navigation tree as you select the editor for that element.

Filtering Displayed Elements

You can filter the Package Navigator view to display only elements of a specified type.

To filter the displayed elements

1. On the Package Navigator toolbar, click .
2. In the **Choose Elements to Display** dialog box, do one of the following:
   - To display all elements, select **Show all elements**.
   - To display only specific types of elements, select **Show selected elements only**. Then, select the elements that you want to display in Package Navigator view.
3. If you want to filter services that Integration Server generates automatically, select the **Hide generated flow services** check box.
4. Click **OK**.

Hiding or Displaying Automatically Generated Flow Services

When you create a step in a process using the Process Development perspective, Integration Server automatically generates flow services for each step. You can instruct Designer to hide or display these automatically generated flow services in Package Navigator view.

To hide or display automatically generated flow services

1. In Designer, click **Window > Preferences**.
2. In the **Preferences** dialog box, select **Software AG > Service Development > Package Navigator**.
3. To hide flow services automatically generated by Integration Server, select the **Hide generated flow services** check box. To show automatically generated flow services, clear the **Hide generated flow services** check box.
4. Click **OK**.

Creating Working Sets

A working set is a subset of elements on one or more Integration Servers. You can create a working set to limit the contents of Package Navigator to only those elements you want to see and work with.

To create a working set

1. On the Package Navigator toolbar, click ▽ and select **Select working set**.
2. In the **Select Working Set** dialog box, click **New**.
3. In the New Working Set dialog box, select **Integration Server Elements.** Click **Next**.
4. In the **Working Set Name** field, enter a name for the working set.
5. Under **Working Set Content**, select the elements to add to the working set. Click **Finish**.
6. If you want to use the new working set, in the **Select Working Set** dialog box, select the working set and click **OK**. Otherwise, click **Cancel**.

Caching Elements

You can improve performance in Designer by caching Package Navigator elements that are frequently used. When elements are located in the Designer cache, Designer does not need to request them from the Integration Server and can therefore display them more quickly.

Keep in mind that increasing the cache reduces the amount of available memory. If you experience memory problems, consider decreasing the number of cached elements.

To cache elements

1. In Designer, click **Window > Preferences**.

2. In the **Preferences** dialog box, select **Software AG > Service Development > Package Navigator**.

3. In the **Number of elements to cache** field, type the number of elements that you want to cache per Designer session. The total number of cached elements includes elements on all the Integration Servers to which you are connected.

   The minimum number of elements is 0. The default is 100 elements. The higher the number of elements, the more likely an element will be in the cache, which reduces network traffic and speeds up Designer.

4. Click **OK**. The caching settings take effect immediately.

Clearing the Designer Cache

When you clear the Designer cache, you remove Package Navigator view elements from the memory for all servers. The following elements are not removed:

- Flow services with breakpoints. (If you want to clear the flow service from the cache, remove the breakpoint and clear the cache again.)
- Flow services that are currently being debugged (for example, a service that has been stepped into.)
- Unsaved elements.

Keep in mind that the cache is automatically cleared when you close Designer or when you refresh your connection to Integration Server.

To clear the Designer cache

1. In Designer, click **Window > Preferences**.

2. In the **Preferences** dialog box, select **Software AG > Service Development > Package Navigator**.

3. Click **Clear Cache**. All cached elements are removed from memory.

Note:
Clearing cached elements from Designer is different from clearing the contents of the pipeline from Integration Server cache. If you want to clear the contents of the pipeline from a server's cache, see "About Service Caching" on page 166.

Exporting Elements

Folders or elements in a package, can be exported to your hard drive so that they can be shared with partners or developers. A folder or element is exported to a ZIP file and saved on your hard drive. The ZIP file can then be unzipped into the ns directory of a package on the server. Locking information is not exported.

To export an element or folder

1. In Package Navigator view, select the folder or element that you want to export to your hard drive.

2. Right-click the element or folder and click **Export from Server**.

3. In the **Save As** dialog box, select the location on your hard drive where you want the exported element or folder to reside. Click **Save**.

   This exports the element or folder to a ZIP file and saves it on your hard drive. The ZIP file can then be published on another server.

Note:
The **Export from Server** option is not the same as the **File > Export** option. With **File > Export**, you can export files from the Workbench to the file system.

Viewing Server Files for an Element

You can view the names of the server files associated with every Integration Server element. This is convenient when an element is system locked and you need to convey the element's file names to the server administrator.

To view server files for an element

1. In the Package Navigator view, select the elements for which you want to view the server file names.

2. Right-click the element and click **Lock Properties**.

   The **Lock Contents Results** dialog box shows the server files associated with the element. These server files are system locked (that is, they are not writable on the server).

Using Property Templates with Elements

Element property templates are sets of pre-defined property values for elements. You can create a template for a particular element and apply the template when creating new instances of the element instead of setting the properties each time you create an element. You can create multiple templates for an element type.

Each element has a default properties template associated with it. The default template of an element applies the default property values to the element. You cannot modify default templates. If you do not apply any template to an element, Designer applies the default template to the element. You can reset an element to its default property values by applying the default template.

Note:
You can create property templates for flow, C/C++, and Java services.

Creating Property Templates

To create an element property template

1. In Designer: **Window > Preferences**

2. In the preferences navigation tree, select **Software AG > Service Development > Element Property Templates**.

3. Click **New**. In the New Template dialog, enter the following information:

Field	Description
Name	Name of the template.
Element Type	The element type for which you are creating the template. **Note:** You can create property templates for flow, C/C++, and Java services.
Description	Description of the template.
Template Properties	Next to each property, specify the value you want to use in the template. For each property, Designer displays a default value. You can edit the fields and specify values for the properties. For more information about specifying the properties, see "Properties" on page 953. **Note:** You will not be able to specify values for properties that must be unique for each element such as **Universal name** and **Output template** when defining templates.

4. Click **OK**.

5. In the Preferences page, click **OK**.

Applying Property Templates to Elements

You can apply property templates to existing elements. You can also reset an element to its default property values by applying the default template.

To apply an element property template to an existing element

1. In the Package Navigator view, right-click the element to which you want to apply the property template and select **Apply template**.
2. Do one of the following:
   a. Select the template that you want to apply to the element.
   b. Select **Default** to apply the default template to the element.

Editing Property Templates

You can change the values of the properties specified in an element property template definition by editing it.

To edit an element property template

1. In Designer: **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > Service Development > Element Property Templates**.
3. Select the property template that you want to edit.
4. Click **Edit**.
5. In the Edit Template dialog, enter new values for the properties that you want to change and click **OK**.
6. In the Preferences page, click **OK**.

Deleting Property Templates

To delete an element property template

1. In Designer: **Window > Preferences**

2. In the preferences navigation tree, select **Software AG > Service Development > Element Property Templates**.

3. Select the property template that you want to delete.

4. Click **Remove**.

5. In the Preferences page, click **OK**.

Note:
When you delete a template, the elements that use the deleted template will be reset to use the default template.

Importing Property Templates

You can share the templates across different instances of the Designer, through import and export operations.

To import an element property template

1. In Designer: **Window > Preferences**

2. In the preferences navigation tree, select **Software AG > Service Development > Element Property Templates**.

3. Click **Import**.

4. In the **Open** dialog, navigate to and select the .xml template file you want to import.

5. Click **OK** to import the template. The template is added to the templates list in the **Preferences** window.

Exporting Property Templates

You can export property template definitions to XML files so that they can be shared across different instances of the Designer.

To export an element property template

1. In Designer: **Window > Preferences**

2. In the preferences navigation tree, select **Software AG > Service Development > Element Property Templates**.

3. Click **Export**.

4. In the **Save As** dialog box, navigate to the folder where you want to save the exported template definition and enter a file name. Click **Save**.

   This exports the template definition to a .xml file.

4 Assigning and Managing Permissions for Elements

- What Is an ACL? 80
- Assigning ACLs 82
- Viewing ACL Information for a Server 84
- ACLs and Locking 84
- ACLs and Running/Debugging Services 85
- ACLs and Creating, Viewing, and Deleting Elements 85
- Troubleshooting ACL Usage 85

You can limit access to elements to groups of users by using access control lists (ACLs). Typically created by a system administrator, ACLs allow you to restrict access on a broader level. For example, if you have a production package and a development package on the Integration Server, you can restrict access to the production package to users in an Administrators ACL, and restrict access to the development package to users in a Developers ACL.

Within ACLs, you can also assign different levels of access, depending on the access that you want different groups of users to have. For example, you may want a "Tester" ACL to only have Read and Execute access to elements. Or, you may want a "Contractor" ACL that denies List access to sensitive packages on the Integration Server, so that contractors never see them in Designer.

What Is an ACL?

An ACL controls access to packages, folders, and other elements (such as services, IS document types, and specifications) at the group level. An ACL identifies groups of users that are allowed to access an element (Allowed Groups) and/or groups that are not allowed to access an element (Denied Groups). When identifying Allowed Groups and Denied Groups, you select from groups that you have defined previously.

There are four different kinds of access: List, Read, Write, and Execute.

- **List** controls whether a user sees the existence of an element and its metadata; that is, its input and output, settings, and ACL permissions. The element will be displayed on screens in Designer and the Integration Server Administrator.
- **Read** controls whether a user can view the source code and metadata of an element.
- **Write** controls whether a user can update an element. This access also controls whether a user can lock, rename, or delete an element or assign an ACL to it.
- **Execute** controls whether a user can execute a service or a web service descriptor.

For more details about these types of access, see *webMethods Integration Server Administrator's Guide*.

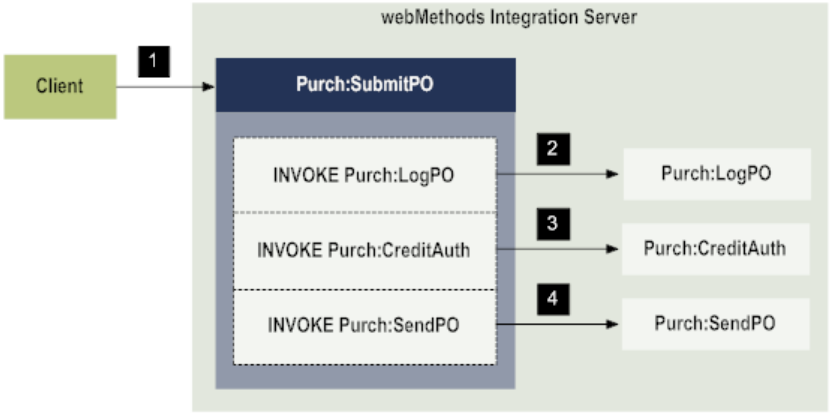
What Happens When a Client Runs a Service with ACLs?

When a client requests that Integration Server invoke a service, the server checks the ACL assigned to the service. If the client is a member of an allowed group *and* is not a member of a denied group, the server executes the service. If the client is not a member of an allowed group, the server denies the request to invoke the service and stops executing.

By default, when a client requests a service, Integration Server checks only the ACL of the externally invoked service (the service requested directly by the client). The server does not check the ACLs of any of the internally invoked services (those services invoked by the externally invoked service). However, you can set up the security settings for a service so that Integration Server checks the ACL assigned to the service *every* time it is invoked, whether directly by a client or by another service. For details, see "Assigning ACLs" on page 82.

The following diagram illustrates the points at which ACL checking occurs when a client requests a service.

ACL checking when a client requests a service



Stage	Description
1	The client (such as another application or a DSP) requests the Purch:SubmitPO service on the local webMethods Integration Server. Integration Server checks the ACL of the Purch:SubmitPO service (the externally invoked service). The server executes the service only if the client is invoking the service on the behalf of a user that is a member of an allowed group and is not a member of a denied group for the ACL assigned to the service.
2	The Purch:SubmitPO service invokes the Purch:LogPO service. Because the Purch:LogPO service is invoked by the externally invoked service and is located on the same server as the externally invoked service, Integration Server considers the Purch:LogPO service to be internally invoked. Consequently, the server does not check the ACL of the Purch:LogPO service before executing it.
3	The Purch:SubmitPO service invokes the Purch:CreditAuth service. Like the Purch:LogPO service, Integration Server considers the Purch:CreditAuth service to be an internally invoked service. Consequently, the server does not check the ACL of the Purch:CreditAuth service before executing it.
4	The Purch:SubmitPO service invokes the Purch:SendPO service. Like the Purch:LogPO and Purch:CreditAuth services, Integration Server considers the Purch:SendPO service to be an internally invoked service. The server does not check the ACL of the Purch:SendPO service before executing it.

Note:
If the security settings for the Purch:LogPO, Purch:CreditAuth, or Purch:SendPO services specify that ACL checking occurs every time the service is invoked (**Enforce execute ACL** option is set to **Always**), Integration Server would perform ACL checking when the externally invoked service (Purch:SubmitPO) invoked these services. For more information about requiring ACL checking, see "Assigning ACLs" on page 82.

Note:
Any service that the Purch:SubmitPO flow service invokes could also be invoked directly by the client. For example, if the client directly invokes the Purch:SendPO service, the server checks the

ACL of the Purch:SendPO service. If the client is invoking the service on the behalf of a user that is a member of an allowed group and not a member of a denied group, then the server executes the Purch:SendPO service.

Is ACL Usage Required?

No. However, there are default ACL settings for elements shipped with Integration Server and default settings for new elements that you create. For details on default ACLs, see *webMethods Integration Server Administrator's Guide*.

Creating ACLs

You create ACLs using the Integration Server Administrator. For details, see *webMethods Integration Server Administrator's Guide*.

ACLs and Inheritance

When you assign an ACL to a folder, it affects the subfolders and services in the folder. The subfolders and services that do not have an assigned ACL inherit the ACLs that you assign to the folder. (Subfolders and services with an assigned ACL are not affected by the ACL assigned to the folder.) When a subfolder or service inherits the ACL of a folder, "**inherited**" is displayed next to the ACL in the **List ACL**, **Read ACL**, **Write ACL**, or **Execute ACL** fields in the Permissions page of the Properties for *elementName* dialog box.

When you remove an ACL from a service or subfolder, the service or subfolder inherits the ACL assigned to the folder in which the service or subfolder is located. When you remove the ACL assigned to the top-level folder (the uppermost folder in a package), Integration Server applies the default ACL to the folder and its contents for which an ACL is not specified. (The default ACL restricts access to a service to any user with a valid user name and password for the Integration Server.)

Default ACLs and Inheritance

If the element is a top-level folder, its default ACL is that specified by a configuration file, not by its parent (the package). If the element is a subfolder, it shares its default ACL settings with other folders at the same level in the folder hierarchy. For details about inheritance, as well as the default ACLs that are installed with the webMethods Integration Server, see *webMethods Integration Server Administrator's Guide*.

Note:
An element can inherit access from all elements except a package.

Assigning ACLs

You can assign an ACL to a package, folder, services, and other elements in the Package Navigator view. Assigning an ACL restricts or allows access to an element for a group of users.

Keep the following points in mind when assigning ACLs:

- You can assign only one ACL per element.
- You can only assign an ACL to an element for List, Read, or Write access if you are a member of that ACL. For example, if you want to allow DevTeam1 to edit the ProcessPO service, you must be a member of the DevTeam1 ACL. That is, your user name must be a member of a group that is in the Allowed list of the DevTeam1 ACL.
- The ACLs assigned to an element are mutually exclusive; that is, an element can have different ACLs assigned for each level of access. For example, the following element has the Developers ACL assigned for Read access and the Administrators ACL assigned for Write access.
- ACL usage is not required. For more information, see "Is ACL Usage Required?" on page 82.

To assign an ACL to an element

1. Make sure that the ACL you want to assign exists on the Integration Server. If not, create the ACL in the Integration Server Administrator. For details, see *webMethods Integration Server Administrator's Guide*.

2. In Package Navigator view, select the package or folder to which you want to assign an ACL and select **File > Properties**. In the Properties for `elementName` dialog box, select **Permissions**.

3. On **Permissions**, select the ACLs that you want to assign for each level of access.

For this permission...	Specify...
List ACL	The ACL whose allowed member can see that the element exists and view the element's metadata (input, output, etc.).
Read ACL	The ACL whose allowed member can view the source code and metadata of the element.
Write ACL	The ACL whose allowed member can lock, check out, edit, rename, and delete the element.
Execute ACL	The ACL whose allowed member can execute the service. This level of access only applies to services and web service descriptors.

4. Under **Enforce execute ACL**, specify when Integration Server performs ACL checking. Select one of the following:

Select...	To specify that...
When top-level service only	Integration Server performs ACL checking against the service when it is directly invoked from a client or DSP. For example, suppose a client invokes the OrderParts service on server A. After checking port access,

Select...	To specify that...
	server A checks the Execute ACL assigned to OrderParts to make sure the requesting user is allowed to run the service. It does not check the Execute ACL when other services invoke OrderParts.
Always	Integration Server performs ACL checking against the service when it is directly invoked from a client as well as when it is invoked from other services. For example, suppose the OrderParts service is invoked from a browser, as well as by the ProcessOrder and AddToDatabase services. If **Always** is set on OrderParts, the server performs ACL checking on OrderParts three times (once when it is invoked from the browser and twice when it is invoked by ProcessOrder and AddToDatabase).

5. Click **OK**.

Viewing ACL Information for a Server

You can view the users and groups that make up the ACLs on a server.

To view ACL information for an Integration Server

1. In Package Navigator view, select the Integration Server.
2. Click **File > Properties**.
3. In the Properties for `serverName` dialog box, select **Server ACL Information.**

   The Server ACL Information page lists the ACLs contained in the Integration Server to which you are connected. This information is read only; to edit ACLs, users, and groups, use the Integration Server Administrator.

ACLs and Locking

As explained previously, locking allows you to control access at the individual user level, while ACLs allow you to control access by groups of users. Following are guidelines to keep in mind as you use ACLs with locking:

- To lock an element, you must be the member of the ACL that is assigned for Write access to that element.
- To lock a Java or C service within a folder, you must be the member of the ACL that is assigned for Write access for *all* Java or C services in that folder. This is because locking and unlocking actions for Java/C services are at the folder level. For details, see "Guidelines for Locking Java and C/C++ Services" on page 90.
- To edit ACL permissions for an element, you must lock the element (except for packages and folders, which cannot be locked).

Note:
When an Integration Server has the VCS Integration feature enabled, an element is locked when it is checked out of the version control system. With the appropriate ACL permissions, you are able to check out (lock) and check in (unlock) elements, folders and packages.

ACLs and Running/Debugging Services

Keep the following in mind when you run and debug services:

- To step through a top-level service, you must have Execute, Read, and List access to the service.
- To step through all the services within a top-level service, you must have Execute, List, and Read access to all services invoked by the top-level service. If you do not have access to services invoked by the top-level service, Designer "steps over" those services. (Integration Server performs ACL checking for a child service when the **Enforce execute ACL** property for the service is set to **Always**.) Designer executes the top-level service and continues to the next flow step. Designer does not step into the top-level service.
- To debug a service by sending an XML file to a service, you must have Read access to the service.
- To set a breakpoint in a service, you must have Read access to the service.

ACLs and Creating, Viewing, and Deleting Elements

Keep the following guidelines in mind when you create, view, or delete elements:

- To create or paste an element, you must have Write access to its parent folder. If you are not a member of the ACL assigned for Write access to the folder, contact your server administrator.
- To copy an element, you must have Read access to it and Write access to its parent folder.
- To rename or delete an element, you must have Write access to it and its parent folder.
- To copy a package, you must be a member of a group assigned to the Replicators ACL.
- When you create a folder and assign an ACL to it, any elements that you create within that folder inherit its ACL, until you explicitly set the ACL to something else. For details about inheritance, see *webMethods Integration Server Administrator's Guide*.
- You may not see all of the elements on the Integration Server in the Package Navigator view because you may not have List access to all of them. You can only see those elements to which you have at least List access.

Troubleshooting ACL Usage

I receive a "Cannot perform operation without Write ACL privileges" message when I try to create an element.

You are not a member of the ACL assigned to the folder in which you want to save the element. To verify, check the Permissions specified in the Properties for *elementName* dialog box. If you had previously been able to save the element, the ACL settings may have changed on the server

since you last saved it. For more information, see "Troubleshooting" on page 94 section in "Locking and Unlocking Elements" on page 87.

I receive an "element already exists" message when I try to create an element.
There may be an element with the same name on the Integration Server, but you may not be able to see it because you do not have List access to it. Try a different element name, or contact your server administrator.

I can't assign an ACL to an element.
Make sure that you have locked the element and that you are a member of the List, Read, or Write ACL that you want to assign. To verify, select the Integration Server and click **File > Properties**. In the Properties for *serverName* dialog box, select **Server ACL Information**. Integration Server. The Server ACL Information page lists the ACLs contained in the Integration Server to which you are connected. This information is read only; to edit ACLs, users, and groups, use the Integration Server Administrator.

I can't see the source of a flow or Java service. However, I can see the input and output.
You do not have Read access to the service. Contact your server administrator to obtain access.

I receive an exception when I try to lock an element.
The element may be locked by someone else, system locked (marked read only on the server), or you may not have Write access. Refresh the Package Navigator view. If a lock is not shown but you still cannot lock the element, reload the package. In addition, make sure that you are a member of the ACL assigned for Write access to the element. To verify, select the element and click **File > Properties**. Select **Permissions** in the Properties for *elementName* dialog box.

I receive an error when I debug a service.
You must have a minimum of Read access to step through a service. If you don't have Read access to the service when you are stepping through, or stepping into a service, you will receive an error message.

If you do have Read access to a service but you do not have Read access to a service it invokes, Designer "steps over" the invoked service but does not display an error message.

You must also have Read access to a service to set a breakpoint in the service or send an XML document to the service.

I receive an exception when I try to go to a referenced service from the pipeline.
You do not have List access to the referenced service. Contact your server administrator.

I receive a "Couldn't find in Package Navigator view" message when I try to find a service in the Package Navigator view. However, I know it is on the Integration Server.
If you do not see the service listed in the Package Navigator view, you probably do not have List access to that service. Contact your server administrator.

I can't copy and paste a Java service.
Check to make sure that you have Write access to all Java services in the folder into which you want to paste the service, as well as Write access to the folder itself.

5 Locking and Unlocking Elements

- What Is a Lock? 88
- About Locking Elements 89
- Viewing the Status of Locked Elements 91
- Copying, Moving, or Deleting Locked Elements 92
- Unlocking Elements 92
- Troubleshooting 94

In Software AG Designer, you can manage changes to elements during development by locking them. This prevents two different users from editing an element at the same time. You can lock elements such as flow services, Java services, schemas, and specifications.

All elements in Designer's Package Navigator view are read-only until you lock them. You can edit an element only if you own the lock on the element. However, you can use and run a service regardless of its lock status, as long as you have Execute access to the service.

Local locking on Integration Server is the default locking mode of Integration Server and Designer.

If you enable Designer's VCS Integration feature, elements are locked and unlocked when you check them out of or into your version control system repository. When an Integration Server has the VCS Integration feature enabled, system locking is effectively disabled for elements that are checked into the version control system. The VCS Integration feature will override any read/write status changes applied manually by a server administrator. For more information about implementing the VCS Integration feature, see *Configuring the VCS Integration Feature* in the *Software AG_directory* _documentation directory.

Note:
If you are using the local service development feature in Designer, the locking mode must be set to system. To do so, set the watt.server.ns.lockingMode parameter to `system`.

What Is a Lock?

A lock on an element prevents another user from editing that element. There are two types of locks: *user locks* and *system locks*. When an element is locked by you, you have a *user lock*. The element is read-only to all other users on the Integration Server. Another user cannot edit the element until you unlock it.

When an element's supporting files (node.xml, for example) are marked read-only on the Integration Server, the element is *system locked*. For example, the server administrator has the ability to mark an element's supporting files on the server as read-only, in which case they are system locked. To edit the element, you must ask the server administrator to remove the system lock (that is, make the element's files writable), and then you must reload the package in which the element resides.

Elements are shown in the following ways in Designer's Package Navigator:

Element	Status	Can I edit?	How do I gain rights to edit?
	Not locked	No	Right-click the element and then select **Lock**.
	Locked by you	Yes	N/A
	Locked by another user	No	Contact the user to unlock.
	Locked by the system	No	Contact the server administrator to unlock.

About Locking Elements

Before you edit an element, you must lock it. Locking ensures that you are the only person working on a particular element at a time, preventing the loss of changes. Elements can only be locked by one user at a time. If the element you need is already locked, request that the current owner of the lock release it. If the element is system locked, request that the server administrator release it by making the corresponding server files writable.

Elements are locked by webMethods user name (the name you use to log on to the Integration Server). Because of this, it is important that you use a distinct user name to log on to the server. If you change user names, you will be unable to edit or unlock items that you locked using your old user name.

Locking Elements in Designer

When locking elements, keep the following points in mind:

- When you create a new element, it is locked automatically for you.
- In order to lock an element, you must have Write access rights to it. For details, see "Assigning and Managing Permissions for Elements" on page 79.
- When you lock an element, Designer obtains and locks the latest version of the element that has been saved on the webMethods Integration Server.
- Elements generated by a service (including an adapter service) are not locked automatically.
- When you select multiple elements to lock, some elements in the selection may not be available to lock because they may be system locked, locked by another user, elements to which you do not have Write access, or elements that cannot directly be locked. Designer will notify you that these elements cannot be locked and will lock the rest.
- When you lock an adapter notification, Designer also locks its associated publishable document type. You cannot directly lock the publishable document type associated with an adapter notification.
- When you lock a folder or package, you only lock existing, unlocked elements within it. Other users can still create new elements in that folder or package.
- When you lock a Java or C/C++ service, Designer locks all other Java or C/C++ services within the folder. This means that other users cannot create Java and C/C++ services in a folder or package that contains the Java or C/C++ services. To create these services, all existing Java and C/C++ services in the folder must be unlocked and the user must have Write access to all Java and C/C++ services in the folder. For details, see "Guidelines for Locking Java and C/C++ Services" on page 90.
- You cannot lock a listener or connection element.

To lock an element

1. In the Package Navigator or in the editor, select the elements that you want to lock.

2. Right-click the elements and then click **Lock**.

   If the elements were successfully locked, a green check mark appears next to their icons in Package Navigator view. If one or more of the elements could not be locked (for example, if they are system locked, locked by another user, or elements to which you do not have Write access), Designer displays a dialog box listing them. For information about troubleshooting lock problems, see "Lock and Unlock Problems" on page 94.

Guidelines for Locking Java and C/C++ Services

When you lock Java and C/C++ services, there are special considerations to keep in mind.

- **Locking and unlocking actions on Java and C/C++ services are folder-wide.** All Java and C/C++ services in a folder share the same .java and .class files on the Integration Server. These files, located in the `\code` subdirectory of a package, correspond to all services (except flow services) in a folder. Therefore, when you lock a Java/C service, all Java/C services in that folder are locked.

  For example, if you lock a Java service in a folder A, all Java and C/C++ services in folder A are locked by you. Similarly, if another user has locked a Java service in folder B, you cannot add, edit, move, or delete any Java or C/C++ services in folder B.

- **Locking actions on Java and C/C++ services are ACL dependent.** If you want to lock one or more Java or C/C++ services within a folder, you must have Write access to all Java and C/C++ services in that folder. This is because Java and C/C++ services within a folder share the same .java and .class files.

- **The jcode development environment operates independently of locking.** If you use jcode to develop Java services, you do not have the locking functionality that is available in the Integration Server. When you use jcode, you may compile a service that is locked by another user, overwriting that user's changes to the service. Therefore, if you use jcode, do not use the locking features in the Integration Server.

- **Before you save a Java or C/C++ service, multiple corresponding files must be writable on the server.**A single Java or C/C++ service corresponds to the following files:

  `.java`
  `.class`
  `.ndf`
  `.frag` (may not be present)

- Before you save a Java or C/C++ service, all of the preceding files must be writable. Therefore, make sure that all system locks are removed from those files before saving.

Guidelines for Locking Templates

A template can be used with one or more services on the Integration Server. Currently, you cannot lock a template as an entity, only the service to which it is attached. Following are considerations for working with templates in a cooperative development environment.

- **To create or edit a template for a service,** you must have the service locked.
- **The template for a service can change without your knowledge.** Since a template can be attached to one or more services, keep in mind that a shared template can change without your knowledge. For example, if your template is attached to a service that another user locks and edits, that user can change your template.

System Locking Elements

If you are a server administrator, you can system lock an element by using the server's file system to make the element's supporting server files read-only. If you do not know the names of the files that correspond to a particular element, use the **Lock Properties** command from the right-click menu. For details, see "Viewing Lock Status of Elements" on page 92. Usually, a system lock is not reflected in Software AG Designer or the Integration Server Administrator until you reload the package in which the element resides.

Important:
Before you system lock an element, always verify that it is not locked by a user on the Integration Server. If an element becomes system locked while a user is editing it, the user will not know until he or she tries to save changes to the element. If this occurs, make the element's corresponding files writable on the server. After this is done, the user can save his or her changes to the element.

Note:
System locking is not supported if you are running webMethods Integration Server as root on a Unix system.

Viewing the Status of Locked Elements

The lock status of an element tells you whether an element is available for locking. If an element is locked, the lock status tells you who owns the lock and when they locked it.

When viewing an element's lock status, keep the following points in mind:

- If the element has been system locked since you last reloaded the package, Designer will not show the system lock status in the Lock Properties dialog box until you reload the package.
- When another user unlocks an element, you must refresh the Package Navigator to reflect the updated status. Similarly, when the server administrator removes a system lock from an element, you must reload the package in which the element resides to reflect the updated status.

Viewing Lock Status of Elements

You can view the lock status of an element in Package Navigator view.

To view the lock status of an element

- In Package Navigator view, right-click the element for which you want to view the status, and then click **Lock Properties**.

  The Lock Contents Results dialog box displays the following information about the locked element:

  - The person who owns the lock on the element.
  - The host on which the locked element resides.
  - The date the element was locked.
  - A list of server-side files that are part of the element.

Listing All of Your Locked Elements

You can view a list of all the elements that are locked by you in the Package Navigator view.

To list all of your locked elements

1. In Package Navigator view, select the server for which you want to view your locked elements.
2. Click **File > Properties > My Locked Elements**.

   You can unlock individual elements from the My Locked Elements page by pressing the CTRL key as you click each element and then clicking **Unlock**. You can unlock all elements by clicking **Unlock All**. For more information about unlocking elements, see "Unlocking Elements" on page 92.

Copying, Moving, or Deleting Locked Elements

You can copy a locked element to another folder or package. However, you cannot move, rename, or delete an element unless it is locked by you or unlocked.

Unlocking Elements

After you edit an element and save changes to the server, you should unlock it to make it available to other users. There are several ways to unlock elements, depending on whether you are a member of the Developers ACL or the Administrators ACL. If you are a developer, you can unlock elements in Designer. If you are an administrator, you can unlock elements in the Integration Server Administrator as well as in Designer.

Unlocking Elements in Designer

You must explicitly unlock elements. Disconnecting from the server does not unlock your element(s), because locks are maintained from session to session.

When unlocking elements, keep the following points in mind:

- Save changes to the elements before you attempt to unlock them.
- When you unlock a single Java or C service, Designer unlocks all other Java or C services within the folder. For details, see "Guidelines for Locking Java and C/C++ Services" on page 90.
- If a Java or C service in a folder has unsaved changes, you will not be able to unlock other Java or C services within that folder. Save the changes and then unlock the services.
- When you unlock an adapter notification, Designer also unlocks its associated publishable document type. You cannot directly unlock the publishable document type associated with an adapter notification.
- You cannot unlock a listener or connection element.

To unlock elements using Designer

1. In Package Navigator view, select the elements that you want to unlock.
2. Right-click the elements and then select **Unlock**.
3. If the elements you want to unlock contain unsaved changes, Designer alerts you that the elements cannot be unlocked until you save the changes. Click **OK** to close the alert dialog box. Then, save the changes and repeat the unlock action.
4. If one of the elements you selected to unlock is a publishable document type associated with an adapter notification, and you did not also select the adapter notification, Designer alerts you that the elements cannot be unlocked. Click **OK** to close the alert dialog box. Then, reselect the elements (including the appropriate adapter notifications) and repeat the unlock action.

   Package Navigator view refreshes and the green check mark next to the element disappears.

Automatically Unlocking an Element Upon Saving

You can choose to automatically unlock flow services, IS document types, and specifications after you save changes to them. This prevents you from forgetting to unlock them; however, it may not be the best option if you save periodically while editing an element.

Important:
When an Integration Server has the VCS Integration feature enabled, the **Automatically unlock upon save** preference must be disabled.

To automatically unlock elements after saving

1. In Designer: **Window > Preferences**
2. In the preference navigation tree: **Software AG > Service Development > Package Navigator**
3. Under Preferences, select the **Automatically unlock upon save** check box.
4. Click **Apply** to save your changes. Or click **OK** to save your changes and close the Preferences dialog box.

Troubleshooting

This following sections address common problems that may arise when implementing cooperative development with webMethods components.

Lock and Unlock Problems

The Lock for Edit and Unlock commands are disabled.
Possible causes are:

- The Integration Server to which you are connected may have the `watt.server.ns.lockingMode` property configured to "none" or "system." For details, contact your server administrator.
- You have selected multiple elements to lock or unlock and your selection contains of one or more of the following:
  - A server
  - A folder or package and its contents
  - A package and any other element
  - An adapter notification record

When I try to lock an element, I get an exception message.
The element may be locked by someone else, system-locked (marked read-only on the server), or you may not have Write access. Refresh the Package Navigator. If a lock is not shown but you still cannot lock the element, reload the package. In addition, make sure that you are a member of the ACL assigned for Write access to the element by checking the element's **Permissions** property in the Properties view.

I cannot unlock a Java or C service.
If there is another Java or C service that is locked by another user or system locked in the same folder, then you cannot unlock any Java or C services in that folder. This is because those services share the same .java and .class files on the Integration Server.

I cannot unlock elements since I changed my user name.

You can only unlock elements that you have locked with your current user name for the session. If you have changed user names, log back in to the server with your old user name and then unlock the elements.

If the administrator has deleted your user name, contact him or her to unlock the elements on the server. You can assist the administrator by using the **Lock Status** command to identify the names of the system-locked files on the server that need to be unlocked.

Another user unlocked an element, but it still shows as locked in the Package Navigator view.
If it is a Java or C service, reload the package in the Package Navigator view. If it is any other element, use the **Refresh** command to refresh the Package Navigator.

I receive an "element failed to unlock" message when I try to unlock elements in the Integration Server Administrator.
This indicates that the server files for the locked element were deleted from the server. You need to update the Integration Server Administrator's list of unlocked elements by clicking **Sync to Name Space** on the **Packages >Management > Locked Elements** screen. The **Sync to Name Space** command runs automatically when the server is started or restarted.

Package Management Problems

I can't preserve locking information when I replicate and publish a package.
This is expected behavior and is part of the feature's design. You can, however, preserve system locks (read-only file attributes).

When I disable a package, it does not preserve locking information.
This is expected behavior and prevents conflicts if another package with the same folder and element names gets installed.

Save Problems

When I try to save an element that I have locked, I get an exception message.
During the time that you had the lock, the element became system-locked, its ACL status changed, or a server administrator removed your lock and another user locked the element. If the exception message indicates that a file is read-only, then one or all of the files that pertain to that element on the server are system-locked. Contact your administrator to remove the system lock. After this is done, you can save the element and your changes will be incorporated.

If the exception message indicates that you cannot perform the action without ACL privileges, then the ACL assigned to the element has been changed to an ACL in which you are not an Allowed user. To preserve your changes to the element, contact your server administrator to:

1. Remove your lock on the element.
2. Lock the element.
3. Edit the ACL assigned for Write access to the element, to give you access.
4. Unlock the element.

You can then save your changes to the element.

When I try to save a template, I get an error message.

The template file on the server is read-only. Contact your server administrator to make the file writable.

Other Problems

I can't delete a package.
One of the elements in that package is system-locked (read-only) or locked by another user. Contact your administrator or contact the user who has the element locked in the package.

The webMethods Integration Server went down while I was locking or unlocking an element.
The action may or may not have completed, depending on the exact moment at which the server ceased action. When the server is back up, restore your session and look at the current status of the element.

Frequently Asked Questions

What is the difference between a system-locked element and a read-only element?
None. "System lock" is a term used to denote an element that has read-only files on the webMethods Integration Server. The server administrator usually applies system locks to files (makes them read-only).

Can I select multiple elements to lock or unlock in Package Navigator view simultaneously?
Yes, you can select multiple elements to lock or unlock in the Package Navigator view of Designer, as long as your selection does not contain the following:

- A server
- A folder or package *and* its contents
- A package and any other element
- An adapter notification record

You can also lock or unlock all elements in a package or folder that have not been previously locked/unlocked by right-clicking the package or folder and selecting **Lock** or **Unlock**.

Where is the lock information stored (such as names of elements that are locked, when they were locked, etc.)?
If you are using the VCS Integration feature, lock information is stored internally in Repository version 4 and in the VCS repository. If you are using local service development, lock information is stored in the VCS repository only.

> **Important:**
> It is not recommended that you use locking and unlocking functionality in an Integration Server cluster. Locking information for elements could be inadvertently shared with another Integration Server in the cluster. Use a standalone Integration Server, not a cluster, during development to eliminate these issues.

Should I archive derived files?
Generally, you should not archive derived files such as the .class file that is generated when you compile a Java service.

What happens to the locks on elements when I replicate a package?

Locking information is not preserved when you replicate and publish a package. This is expected behavior and is part of the feature's design. You can, however, preserve system locks (read-only file attributes).

6 Using the Local Service Development Feature

- About the Local Service Development Workflow 101
- How Does Local Service Development Differ from the VCS Integration Feature? 101
- Supported Platforms and Eclipse Plug-ins 103
- Supported Elements 103
- Supported and Unsupported Actions 103
- Prerequisites 104
- Permissions and Locking 105
- Setting the Local Development Server 106
- Creating a Local Service Development Project 108
- Adding Folders and Elements to the VCS 109
- Modifying Packages, Folders, or Elements in the VCS 110
- Checking Out an Element from the VCS 111
- Checking In Packages and Element to the VCS 112
- Getting the Latest Version from the VCS 113
- Getting a Specific Version from the VCS 114
- Copying Packages from the VCS to Integration Server 115
- Reloading a Package 116
- Comparing Revisions of an Element 116
- Building Java and C Services 117

- Deleting a Package Associated with a Local Service Development Project 118
- Deleting a Local Service Development Project 118

The local service development feature is a Designer feature that you can use to develop Integration Server packages locally as Eclipse projects. With this feature, you can check package elements and their supporting files in to and out of a version control system (VCS) directly from Designer.

To connect Designer to a VCS client, the local service development feature uses the following components:

- A local development package, which is an Integration Server package that is intended to be used with the local service development feature.
- A local development project, which is an Eclipse project that contains the associated local development package.
- A local development server, which is an Integration Server instance that is installed in the same installation directory as the Designer instance you are using.

About the Local Service Development Workflow

The typical work flow for using the local service development feature within Designer is as follows:

1. Define a local service development project in the Eclipse workspace and select the VCS client you want to use. For details, see "Creating a Local Service Development Project" on page 108.
2. Select the package, folder, or element that you want to place in your VCS, and then check the selected item in to your VCS repository. For details, see "Adding Folders and Elements to the VCS" on page 109.
3. Check out the package, folder, or element for modification on your local development server. For details, see "Checking Out an Element from the VCS" on page 111 and "Copying Packages from the VCS to Integration Server " on page 115.
4. Modify the item, save your changes, and check the item back in to your VCS repository. For details, see "Modifying Packages, Folders, or Elements in the VCS" on page 110 and "Checking In Packages and Element to the VCS" on page 112.

How Does Local Service Development Differ from the VCS Integration Feature?

The local service development feature is not the same as the VCS Integration feature. The following table describes the differences:

Important:
The WmVCS package, which provides the functionality for using the VCS Integration Feature, is deprecated as of Integration Server version 9.9. Software AG recommends that you use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer.

Characteristic	Local Service Development	VCS Integration
Installation method	Installed as a plug-in to Designer	Provided by way of the WmVCS Integration Server package
Tasks required to support new versions of VCS clients	Reinstall the VCS client plug-in; no Designer upgrade needed	Update the WmVCS package every time a new version of the VCS client is released
Where configured	Designer	Integration Server
Number of VCS clients permitted to connect to	Permits Designer to connect to multiple VCS clients	Permits Integration Server to connect to only one of the supported VCS clients
How version control tasks are performed	Locally, within the Eclipse framework (commands are sent directly between Designer and the VCS client)	Through Integration Server
Menus and commands used	Uses the VCS client's menus and commands, which may already be familiar to you	Uses its own commands to access the VCS client, which may require extra time to learn
VCS clients supported	■ Concurrent Versions System (CVS) ■ Subversion (SVN) ■ Microsoft Team Foundation Server ■ Eclipse Git Team Provider (EGit)	■ Microsoft Visual SourceSafe ■ IBM Rational ClearCase ■ Subversion (SVN) ■ Exposed Java API that you can use to develop a connection to other VCS clients
Method of communicating with VCS client	Communicates through Eclipse plug-ins that support versions of the clients; plug-ins are either installed by default with Eclipse or available through third-party providers	Communicates directly with the VCS client
Assets permitted to maintain	Permits check-in and check-out of all contents of a package, including Integration Server assets and other items that Integration Server does not maintain, such as HTML files	Requires manual check-in and check-out of items that Integration Server does not maintain

Supported Platforms and Eclipse Plug-ins

The version of the VCS clients supported by the local service development feature depends on the Eclipse plug-ins supported for each VCS client. The local service development feature supports the following Eclipse plug-ins:

VCS	Eclipse Plug-in Version
CVS	CVS 1.4.x
SVN	Polarion Software Subversive SVN plug-in and connector 3.0.x
SVN	Subclipse 4.2.x
Team Foundation Server	Team Foundation Server plug-in for Eclipse Version 14.x
Git	Eclipse Git Team Provider 3.4.x

Supported Elements

The local service development feature works with all of the packages and IS elements displayed in the Package Navigator view, as well as package contents that are not visible in the Package Navigator view, such as supporting files associated with the folder or element.

Note:
The local service development feature works with the local development server only. The feature does not work with packages in other servers listed in the Package Navigator view.

Supported and Unsupported Actions

The local service development feature supports the following actions:

- Create a project and share it with a VCS.
- Check packages, folders, and elements in to and out of a VCS.
- Retrieve a specific version of a package, folder, or element from the VCS repository.
- Copy a package from the VCS repository to your local development Integration Server.
- Reload a package on the local development Integration Server.
- Delete a package associated with a local service development project.
- Delete a local service development project.
- Restore a deleted element from the VCS repository using the VCS client's explorer view.
- View history from the VCS client's explorer view.

The local service development feature does NOT support the following actions:

- Display differences between versions.
- Merge partial changes from different revisions.
- Move, copy, rename, or delete items from the VCS client's explorer views. You must perform these actions in the Package Navigator in Designer.

Prerequisites

Before you use the local service development feature, you must:

- Ensure that Integration Server is installed in the same installation directory as the Designer instance you are using. If you selected **Local Version Control Integration** (or **Designer Workstation** in versions prior to 9.8) from the Software AG Installer, this was already done for you.
- Ensure that the watt.server.ns.lockingMode parameter is set to `system` on the Integration Server used as the local development server. If you set any other value for this parameter, the Local Service Development feature may not work as expected. For more information about this parameter, see *webMethods Integration Server Administrator's Guide*.
- Install the Workstation local service development feature as described in *Installing Software AG Products*.
- Ensure that the VCS Integration feature is not configured on the local development server.
- Install the correct version of the VCS client that you want to use, and if necessary, install the correct Eclipse plug-in for that client. For details about supported Eclipse plug-ins and VCS versions, see "Supported Platforms and Eclipse Plug-ins" on page 103.

  Important:
  If you are using Subversion as your VCS client, Software AG recommends that you use Subversion 1.7 or higher. Using versions of Subversion prior to 1.7 might cause issues while refreshing web service descriptors and web service connectors.

- In Designer, configure your workspace to be refreshed automatically by selecting the **Refresh using native hooks or polling** or **Refresh automatically** check box in **Window > Preferences > General > Workspace**.
- In Designer, remove .BAK and .bak from the list of resource name patterns to exclude from the VCS. To do this, clear the .BAK and .bak check boxes in **Window > Preferences > Team > Ignored Resources**.
- If you are using CVS as your VCS client, in Designer, clear the **Prune empty directories** check box in **Window > Preferences >Team > CVS > Files and Folders**.
- If you are using EGit as your VCS client, clear the **Auto share projects located in a git repository** check box in **Window > Preferences >Team > Git > Projects**.
- If you are using EGit as your VCS client, before creating the local service development project, create a new local Git repository. To create a new local Git repository, select **Create new local Git repository** in the **Git Repositories** view.

You can also use the **Create Repository** field in the Configure Git Repository screen of the Share Project wizard to create and select local Git repository while creating the local service development project.

Clear the **Move project to Integration Server package as linked resource** check box in **Window > Preferences > Software AG > Service Development > Local Service Development** to set the *Integration Server_directory* \instances\default\packages directory as the local Git repository. You can use the local Git repository for all further local development operations.

- If you are using Team Foundation Server as your VCS client, in Designer, clear the **Connect mapped projects to TFS automatically** check box in **Window > Preferences >Team > Team Foundation Server** before sharing the local service development project. Otherwise, Team Foundation Server might throw an exception when sharing the local service development project.
- For local development in Integration Server or Microservices Container running inside a Docker container, before you start the Docker container, ensure that you mount the corresponding host directory of local Git repository or the TFS project directory in the Docker container. Consider the following sample startup command:

```
docker run -i -t -d \
 --name $[container_name] \
 -v $[container_named_volume]: [Install_Dir]/IntegrationServer/instances/
[instance_Name] \
 -v [host_git_project_location]:/[container_git_project_location]
 -p [host_primary_port:]primary_port \
 [image_name] \
 /bin/bash
```

Permissions and Locking

Two mechanisms exist to control access to an element and to avoid editing conflicts:

- Access control lists (ACLs), which grant permission to view and edit an element
- Locking, which assures that only one developer at a time can edit an element

Permissions

Access control lists (ACLs) determine the level of access to packages, folders, and other elements (such as services, IS document types, and specifications) at the group level. ACL settings are stored on the local development Integration Server, not with the elements themselves. This means that ACL information does not get checked in to the VCS repository when you check in an element. When another user checks the element out of the VCS repository, that user's local development server uses the default ACL to determine access to that element. You can preserve ACL settings for an element by deploying the element from the local development server and then setting the element's ACL settings manually on the production server. For more information about ACLs, see "Assigning and Managing Permissions for Elements" on page 79.

System Locking and Local Service Development

Designer and Integration Server support the concept of system locking. When an element's server-side files are marked read-only on Integration Server, the element is system locked. Files are generally system locked when a server administrator accesses the file through the server's operating system and marks the files as read-only. You cannot edit an element until the server administrator makes the element's server-side files writable and you reload the package in which the element resides.

System locking is disabled for elements that are checked into the VCS with the local service development feature. The local service development feature will override any read/write status changes applied manually by a server administrator.

Note:
On the Integration Server used as the local development server, the watt.server.ns.lockingMode parameter must be set to `system`. If you set any other value for this parameter, the Local Service Development feature may not work as expected. For more information about this parameter, see *webMethods Integration Server Administrator's Guide*.

Note:
If you are using Team Foundation Server as your VCS client, Integration Server system locks all elements and marks them as read-only. To unlock an element in preparation for editing it, select the **Check out for edit** option from the **Team** menu.

Setting the Local Development Server

A local development server can be one of the following server instances:

- On-premise Integration Server
- On-premise Microservices Container
- Integration Server running inside a Docker container
- Microservices Container running inside a Docker container

Note:
On-premise Integration Server and on-premise Microservices Container are referred to as Integration Server and Microservices Container respectively in this document, unless specifically stated otherwise.

The local service development feature works with the local development server only.

By default, a new Designer installation includes a single server definition named Default. This server is marked as the default server and is configured to use localhost:5555.

If your Designer installation needs to connect to more Integration Servers, you can configure additional Integration Server instances on the **Window > Preferences > Software AG > Integration Servers** page. If there are multiple Integration Server instances configured, only one instance will be active at a time. The default Integration Server instance will be treated as the default local

development server. However, you can set any of the available local Integration Server instances as the local development server. Perform the following step to set an Integration Server instance as the local development server.

- In Package Navigator view, right-click the Integration Server instance that you want to use as a local development server and select **Use as Local Version Control Integration Server**

  The server icon changes to indicating that it is an active local development server.

  Note:
  The **Use as Local Version Control Integration Server** option is available only if the selected Integration Server is connected and is installed in the same installation directory as the Designer instance.

Setting an Integration Server or Microservices Container as the Local Development Server

In versions prior to 9.10, Designer used only the Integration Server instance named "default" as the local development server. However, from version 9.10 release, you can set any Integration Server instance as the local development server as long as the selected instance is installed in the same installation directory as the Designer instance you are using.

Setting an Integration Server or Microservices Container Running inside a Docker container as the Local Development Server

To setup Integration Server or Microservices Container running inside a Docker container as local development server, ensure the following prerequisites are met:

- Docker and Designer reside on the same host.
- Docker daemon is up and directly listens to the Docker Remote API requests using a TCP socket without authentication and decryption.
- Root storage area of the server instance in the Docker container is available on the host.

  To do this, set `Dtarget.configuration=localdev` when creating a `Dockerfile` to build the Docker image for Integration Server or Microservices Container. This configuration creates a mount point using the volume instructions in the `Dockerfile`. For information on creating a Docker file and Docker image, see *webMethods Integration Server Administrator's Guide*.

- The volume is mounted as named volume while starting up the container.

  For example, consider the following sample container startup command:

```
docker run -i -t -d \
 --name $[container_name] \
 -v $[container_named_volume]:
[Install_Dir]/IntegrationServer/instances/[instance_Name] \
 -p [host_primary_port:]primary_port \
 [image_name] \
 /bin/bash
```

Where...	Specify...
container_name	Name of the Integration Server or Microservices Container in the Docker for running the Integration Server orMicroservices Container image. For example, IS_Default.
container_named_volume	Named volume of the Docker.
Install_Dir	Installation directory of Integration Server or Microservices Container.
host_primary_port :	A port on the container host machine that you want to explicitly map to an exposed port on the Integration Server or Microservices Container, specifically the primary port.
primary_port	Port number for the primary port on Integration Server orMicroservices Container.
image_name	Name of Integration Server or Microservices Container image. For example, is:microPkg.

Creating a Local Service Development Project

To use the local service development feature in Designer, you must first define a project in the workspace that represents a package in Integration Server. By creating the local service development project and adding it to the VCS, you can share your files using your VCS client. When you create a project for a package, the package along with its contents are added to the VCS.

Keep the following points in mind when creating a local service development project:

- The package for which you want to create a local development project must exist and be located in the *Integration Server_directory* \instances\default\packages directory folder of the local development Integration Server.
- The package for which you want to create a local development project must not be present in the VCS.
- A project with the same name as the package must not be present in your Eclipse workspace.

To create a local service development project

1. In Package Navigator view, right-click the package for which you want to create a local service development project and select **Create Local Service Development Project**.

   Designer creates the local service development project and displays the Share Project wizard.

2. From the list of VCS client plug-ins, select the VCS client that you want to use to share the new project. Click **Next**.

3. The screens that appear next are specific to the VCS client that you selected. Enter the relevant information and credentials of your VCS client.

4. In the confirmation screen, ensure that the displayed information is correct and click **Finish**.

   Designer creates a project in your workspace with the same name as the package.

   In Package Navigator view, the icon representing the package that you have shared to the VCS changes to showing that the package is shared. The package and the elements contained in the package will now be available in the VCS. Designer displays icon overlays that are specific to your VCS for the files in the shared package in Package Navigator view. These icon overlays indicate the VCS status of the files in your workspace.

5. If you are using Team Foundation Server as your VCS client, you must do the following after the project is created:

   a. Add the content of the project to the Team Foundation Server repository. To do this, right-click the project in Package Explorer view or Navigator view and select **Team > Check In Pending Changes**.

   b. Set the Team Foundation Server working folder to any system location. To set the *Integration Server_directory* \instances\default\packages directory as the working directory, clear the **Move project to Integration Server package as linked resource** check box in **Window > Preferences > Software AG > Service Development > Local Service Development**. To do this, from the Team Explorer view, open the Source Control editor. Right-click the local service development project in the Source Control editor, and select **Set Working Folder**

Adding Folders and Elements to the VCS

Keep the following points in mind when you add a folder or element to a VCS repository:

- When you add a folder, all of the supported elements within the folder's hierarchy are added to the VCS repository.
- When you add a folder or an element in a folder, all of the folders in the path to the folder or element will also be added to the VCS repository. However, the remaining contents of the package, as well as folders that are not part of the path to the element, will not be added to the repository.
- When a folder containing coded services (Java and C/C++ for example) is added to the VCS repository, all of the coded services in that folder will be added to the repository when you check in or commit the folder.
- You must save the changes, if any, to the package, folder, or element before adding it to the VCS repository. If not, your unsaved changes will not be reflected in the VCS.
- When you create a local service development project for a package, any folders or elements contained in the package at the time will also be available in the VCS. Any element that you

add to the package after you create the local service development must be specifically added to the VCS.

To add a folder or element to the VCS

- Perform one of the following actions:
  - In Package Navigator view, right-click the folder or element that you want to add to the VCS. From the context menu, select **Team**, and select the check in or commit option specific to your VCS client. To select multiple folders or elements, press the CTRL key while selecting.

    --OR--

  - In Package Navigator view, right-click the folder or element and select **Show Files**. Designer highlights all the server files associated with the folder or element in the Navigator view. Right-click these files in Navigator view, select **Team**, and select the check in or commit option specific to your VCS client.

Note:
You can also add any file that is outside the package namespace structure or that is not an IS asset (that is, it does not appear in the Package Navigator view of Designer). For example, an html file (output template files for a service) in the pub folder of a package. To do this, right-click the files in Navigator view, select **Team**, and select the check in or commit option specific to your VCS client.

Modifying Packages, Folders, or Elements in the VCS

Keep the following points in mind when you modify elements in a VCS repository:

- If you check out a package or a folder, Designer checks out all of the folders and elements within the package or folder, along with their supporting files.
- If you check out a service, Designer checks out the node.ndf file for that service. Depending on the service type, Designer also checks out other supporting files such as flow.xml and java.frag files.
- When you edit elements locally and check the elements back in to the VCS repository, Designer informs you of any possible conflicts that exist between your local version and the version in the repository. Depending on the VCS client you are using, the client may either merge the changes so they are reflected both locally and in the repository, or prompt you to decide the appropriate action to resolve the conflict.
- If your VCS client is Team Foundation Server, you must first check out the asset before modifying it.
- If your VCS client is SVN or CVS:
  - If the element you want to modify already exists in your *Integration Server_directory* \instances\default\packages directory, you can modify the element without checking it out.

- If the element you want to modify does not exist in your *Integration Server_directory* \instances\default\packages directory, you must first check out the element from the VCS repository. Then, you must move the package to the *Integration Server_directory* \instances\default\packages directory of the Integration Server assigned to be the local service development server. For information about copying packages from VCS, see "Copying Packages from the VCS to Integration Server " on page 115.

- The changes that you made to the package, folder, or element are reflected in the VCS only after you commit your changes to the VCS.

To modify a package, folder, or element that is in the VCS

1. In Designer, in the VCS repository view specific to your VCS client, right-click the package, folder, or element whose contents you want to modify.

2. From the **Team** menu, select the appropriate option to check out the files. The options available in the **Team** menu depend on the VCS client that you use.

   If you are checking out a package or a folder, all the contents of the package or folder also gets checked out. The element(s) that are checked out are now available for editing in your Eclipse workspace.

3. After you complete your modifications, in Package Navigator view, right-click the checked-out package, folder, or element and check in or commit these files to the VCS using the option that is specific to your VCS client, available in the **Team** menu.

   Note:
   You can also right-click the checked-out package, folder, or element and select **Show Files**. You can select multiple folders or elements by pressing the CTRL key while selecting. Designer highlights all the server files associated with the package, folder, or element in the Navigator view. Right-click these files in Navigator view and check in or commit these files to the VCS using the option that is specific to your VCS client, available in the **Team** menu.

   If you are checking in a package or folder, all the contents within it are also checked in.

Checking Out an Element from the VCS

Keep the following points in mind when checking out an element from the VCS:

- The package, folder, or element that you want to check out must exist in the VCS repository.
- When you check out a package or a folder, all the contents of the package or folder are also checked out.
- After the check-out procedure is completed, a small check mark or '>' mark appears on the icon of the checked-out element in the Navigator view.
- If you are using Team Foundation Server as your VCS client:

- When you check out a package, it is added to the directory which is set as the working folder.
- If an element or a package with the same name already exists in the *Integration Server_directory* \instances\default\packages directory, Designer will merge the two elements or the contents of the two packages.
- After checking out, you must right-click the package in Navigator view and select **Load IS Package** to load the package in the *Integration Server_directory* \instances\default\packages directory.

- If you are using CVS or SVN as your VCS client, if a package or an element with the same name already exists in the *Integration Server_directory* \instances\default\packages directory, Designer replaces the package or the element with the one from the VCS.
- If **Move project to Integration Server package as linked resource** is selected, you can clone remote Git repository to any location on your system. Then, you must import the required projects to the workspace to view them in Package Explorer view or Navigator view. You must also load the package(s) in the *Integration Server_directory* \instances\default\packages directory by right-clicking the package in Navigator view and selecting **Load IS Package** and **Move Project to IS Package**.

To check out a package, folder, or element from the VCS

1. In Designer, in the VCS repository view specific to your VCS client, right-click the package, folder, or element that you want to check out. To select multiple folders or elements, press the CTRL key while selecting.

2. From the **Team** menu, select the appropriate option to check out the files. The options available in the **Team** menu depend on the VCS client that you use.

   Designer displays the Progress Monitor dialog box. Click **Run in Background** to continue working in Designer.

   Important: Software AG recommends that you do not perform any operation on the asset on which Designer is performing the VCS operation because it might result in the asset becoming corrupted.

   The element(s) that are checked out are now available for editing in your Eclipse workspace.

   Note:
   If you are checking out a package that you have not checked in previously, you must move this package from your workspace to the *Integration Server_directory* \instances\default\packages directory of your local service development Integration Server.

Checking In Packages and Element to the VCS

Keep the following points in mind when checking in a package, folder, or an element to the VCS:

- When you check in a package or a folder, all the contents of the package or folder are also checked in.
- If a folder or element does not exist in the VCS, checking it in adds the folder or element to the VCS repository.
- If a package, folder, or element has unsaved changes, you must save the changes before checking it in. If not, your unsaved changes will not be reflected in the VCS.
- When checking in a Java or C/C++ service, you must check in from the package level so that all files associated with the Java or C/C++ service are also checked in.
- After the check in procedure is completed, the small check mark or '>' mark that indicates that the element is checked out is removed from the element icon in the Navigator view.

To check in a package, folder, or element to the VCS

1. In Designer, in Package Navigator view, right-click the package, folder, or element that you want to check in and from the **Team** menu, select the appropriate option to check in the files. The options available in the **Team** menu depend on the VCS client that you use.

   Note:
   You can also right-click the package, folder, or element that you want to check in and select **Show Files**. You can select multiple folders or elements by pressing the CTRL key while selecting. Designer highlights all the server files associated with the package, folder, or element in the Navigator view. Right-click these files in Navigator view and, from the **Team** menu, select the appropriate option to check in the files.

2. Designer displays the Progress Monitor dialog box. Click **Run in Background** to continue working in Designer.

   Important: Software AG recommends that you do not perform any operation on the asset on which Designer is performing the VCS operation because it might result in the asset becoming corrupted.

   Note:
   You can also check in any folders or elements that are outside the package namespace structure or files that are not IS assets (that is, they do not appear in the Package Navigator view of Designer). To do this, right-click the files in Navigator view, select **Team**, and select the check in or commit option specific to your VCS client.

Getting the Latest Version from the VCS

Use the following procedure to replace packages, folders, and elements on the Integration Server used as the local service development server with the most recent version in the VCS repository.

To get the latest version of a package, folder, or element from the VCS

1. Close the respective editor if you have any of the elements on which you are performing the get latest operation open in the Package Navigator view.

2. In Package Navigator view, right-click the package, folder, or element for which you want to retrieve the latest version and select **Team > Get Latest Version**.

   Note:
   The menu option **Get Latest Version** might differ depending on the VCS client that you use.

   Designer displays the Progress Monitor dialog box. Click **Run in Background** to continue working in Designer.

   Important: Software AG recommends that you do not perform any operation on the asset on which Designer is performing the VCS operation because it might result in the asset becoming corrupted.

   Designer retrieves the latest version of the package, folder, or element and displays a confirmation message.

   Note: Designer automatically rebuilds the Java and C/C++ services in a local service development package, when you get the latest version of the package from the VCS, if you have the **Build Automatically** workspace preference (**Project > Build Automatically**) selected.

Getting a Specific Version from the VCS

Use the following procedure to replace packages, folders, and elements in Integration Server with an earlier version in the VCS repository.

To retrieve a specific version from the VCS, you must know the VCS version number, date, workspace version, or VCS label of the earlier version of file depending on the options that your VCS client provides. You can obtain this information by right-clicking the package, folder, or element in the Navigator view and selecting **Team > View History** or similar option that is specific to your VCS client.

Note: Designer automatically rebuilds the Java and C/C++ services in a local service development project or package, when you retrieve a specific version of the project or package from the VCS, if you have the **Build Automatically** workspace preference (**Project > Build Automatically**) selected.

To get a specific version of a package, folder, or element from the VCS

1. Close all open editors in the Package Navigator view.

2. In Package Navigator view, right-click the package, folder, or element for which you want to retrieve a specific version, and select **Team > Get Specific Version** or similar option that is specific to your VCS client.

Copying Packages from the VCS to Integration Server

When you check out a package that you have not checked in previously, the package is available in your project workspace. You can move this package from your workspace to the *Integration Server_directory* \instances\default\packages directory of your local service development Integration Server using the following procedure.

Keep the following points in mind when copying packages from the VCS to Integration Server:

- You must add VCS project nature to the packages that are not created in the local Integration Server but are available in the VCS if you want to copy or move these packages to Integration Server. To do this, right-click the package in the Package Explorer view and select **Configure > Convert to Local Service Development Project**. This also adds the Java project nature to the package.
- If you are using Team Foundation Server as your VCS client, the workspace of the local service development project is set to the *Integration Server_directory* \instances\default\packages directory. When you check out a package, it is added to the *Integration Server_directory* \instances\default\packages directory. Hence, you will not see the **Move Project to IS Package** option described below. However, you can load the package in the *Integration Server_directory* \instances\default\packages directory by right-clicking the package in Navigator view and selecting **Load IS Package**. After you load the package, the **Load IS Package** option will change to **Reload IS Package**.
- If you are using EGit as your VCS client, select **Move project to Integration Server package as linked resource** to set the workspace of the local service development project to any directory on your system and check out packages to that directory.
- If a package with the same name already exists in the *Integration Server_directory* \instances\default\packages directory, the behavior of Designer depends on your VCS client:
  - If you are using Team Foundation Server as your VCS client, Designer will merge the contents of the two packages.
  - If you are using CVS or SVN as your VCS client, Designer will replace the existing package with the package from the VCS repository.
- Designer automatically builds the Java and C/C++ services when you move the associated project to the *Integration Server_directory* \instances\default\packages directory, if you have the **Build Automatically** workspace preference (**Project > Build Automatically**) selected.

To copy a package from the VCS repository to Integration Server

1. From your VCS repository, get the latest version of the package that you want to copy to your *Integration Server_directory* \instances\default\packages directory.

   Designer copies the package to your workspace.

2. In Navigator view, right-click the package and select **Move Project to IS Package**.

The package is now available in the *Integration Server_directory* \instances\default\packages directory.

If the package that you copy from the VCS repository is not enabled, Designer displays a message informing you that the package is in a disabled state. You can use Integration Server Administrator to enable the package. For more information about enabling a package, see *webMethods Integration Server Administrator's Guide*.

Reloading a Package

If you make any changes to the package and/or it's contents in your workspace, you must reload the package on the local development server to activate the changes and to make sure that the changes are reflected in the *Integration Server_directory* \instances\default\packages directory.

To reload a package

1. In Navigator view, right-click the package that you need to reload and select **Load IS Package**.

2. If you need to replace the package in the *Integration Server_directory* \instances\default\packages directory, right-click the package in Navigator view and select **Reload IS Package**.

   If the package that you copy from the VCS repository is not enabled, Designer displays a message informing you that the package is in a disabled state. Use Integration Server Administrator to enable the package. For more information about enabling a package, see *webMethods Integration Server Administrator's Guide*.

   Note: Designer automatically rebuilds the Java and C/C++ services in a local service development package, when you reload the package on the local development server, if you have the **Build Automatically** workspace preference (**Project > Build Automatically**) selected.

Comparing Revisions of an Element

You can use the compare tool to compare two revisions of an element in a local service development project.

Important:
Revision compare is currently supported only for document types and flow services that are part of a local service development project.

Keep the following points in mind when you use revision compare:

- The **Compare Element(s) With > Revision** option is available for a document type or flow service in a local service development project only if:
  - You have checked out the element to the Project Explorer; and

- You have reloaded the element on the local development server to make sure that the changes are reflected in the *Integration Server_directory*\instances\default\packages directory.

- Revision compare is supported for SVN (Polarion Software Subversive SVN) and Git VCS clients.

To compare the local service development project revision of an element with a specified revision of the element

1. In the Package Navigator view, select an element in a local service development project, right-click, and select **Compare Element(s) With > Revision**.

2. In the dialog box that opens, specify the revision of the element that you want to compare with by choosing one of the following options:

   - Select the **Head Revision**.
   - Specify the **Date** and time values corresponding to the revision.
   - Specify a **Revision** number or use the **Browse** button to browse the VCS and select a revision of the element.

3. Click **OK** to display the list of differences in a compare editor. For information on the compare editor, see "Working with the Compare Editor" on page 922.

Building Java and C Services

When you check out a project containing Java or C/C++ services from the VCS repository, the services might not be operational. That is, the projects might not contain the .java and .class files that Integration Server needs to execute the service. Hence, before executing the Java or C/C++ services, you must build those services.

Designer automatically rebuilds the Java and C/C++ services in a local service development package, when you perform any VCS operations on the package or project in the local development server

Before checking out Java or C/C++ services, you must:

- Ensure that the system environment variable PATH is set to include *JDK_directory* \bin.

  --OR--

- Ensure that the watt.server.compile property is set to *JDK_directory* `\javac -classpath {0} -d {1} {2}` in the config.ini file that is located in the *Software AG_directory* \eclipse\v36\configuration directory.

  For example,

```
watt.server.compile=C\:\\softwareag\\jvm\\jvm160_32\\bin\\javac
-classpath {0} -d {1} {2}
```

To ensure that Designer automatically builds the Java and C/C++ services, you must do the following:

- Add VCS project nature to the packages that contain the Java or C/C++ services, if these packages are not created in the local Integration Server but are available in the VCS repository. To do this, right-click the package in the Package Explorer view and select **Configure > Convert to Local Service Development Project**. This also adds the Java project nature to the package.
- Select the Build Automatically workspace preference. To do this, select **Project > Build Automatically**.

Upon building a Java service, Designer creates the .java and .class files. In case of C/C++ services, Designer generates the Java class associated with the C/C++ service.

Deleting a Package Associated with a Local Service Development Project

When you delete a package that is associated with a local service development project, the associated project in your workspace is also deleted.

To delete a package associated with a local service development project

1. In Package Navigator view, right-click a package associated with a local service development project and select **Delete**.

2. Designer displays a confirmation message asking if you want to delete the associated local service development project as well. Click **Yes** to confirm the deletion.

   Designer displays a message stating that the deletion is complete and that the deleted package has been copied to the recovery area of the Integration Server. Click **OK**.

   Note:
   If you delete a package using Integration Server Administrator or a built-in service and if you attempt to perform any operations on the deleted package from Navigator view, Designer will display a message stating that the project is deleted from the local service development workspace. For the project to function correctly, you must restore the deleted packages.

Deleting a Local Service Development Project

If you have already shared a package to a VCS client and you want to move it to another VCS client, you must first disconnect from the current VCS client. You can then delete the local service development project only and not the package associated with it. You can share the package to another VCS client using the procedure given in "Creating a Local Service Development Project" on page 108.

To delete a local service development project

1. In Package Navigator view, right-click a local service development project and select **Team > Disconnect** or **Team > Disconnect Project Permanently**. The menu options depend on the VCS client that you use.

   Important:
   When disconnecting a local service development project from the VCS, any pending changes you have in this project will be lost. If you want your changes to be updated in the VCS repository, make sure that you check in your pending changes to the VCS before disconnecting the project from the VCS.

2. Depending on the VCS client you are using, do the following:

   - For CVS, select **Also delete the CVS meta information from the file system**
   - For SVN, select **Also delete the SVN meta-information from the file system**

3. If a confirmation message appears, click **OK**. The package is now disconnected from the VCS.

   Note:
   If you want to connect to a VCS again, you need to share the project to the VCS by right-clicking the project in Package Explorer view or Navigator view and clicking **Team > Share Project**.

4. To delete the local service development project, right-click the project in Navigator view and select **Delete**.

5. Click **OK** to confirm the deletion of the project.

   The local service development project is deleted, but the associated package will still be available in the Package Navigator view and in the *Integration Server_directory* \instances\default\packages directory.

7 Using the VCS Integration Feature to Check Elements In and Out of a VCS

- VCS Integration Supported Features 123
- VCS Integration Unsupported Features 124
- Locking Locally vs VCS Locking 124
- System Locking and VCS Integration Feature 124
- About Unlocking Elements with Integration Server Administrator 125
- Adding New Packages and Elements to a VCS 125
- Adding Existing Packages and Elements to a VCS 125
- Modifying Elements that are in the VCS 126
- Checking Out Packages and Elements 126
- Checking In Packages and Elements 127
- Reverting Changes to a Checked Out Package or Element 129
- Getting the Latest Version from the VCS 130
- Getting an Earlier Version from the VCS 130
- Deleting Packages and Elements from the VCS 132
- Restoring Deleted Items 133
- Copying and Moving Folders or Elements 135
- Renaming Packages, Folders, and Elements 135
- Viewing the History of a Folder or Element 135

- Working with Blaze Rules Services 137
- Working with Web Service Descriptors 137
- Working with webMethods Adapter Connections 137
- Working with Java Services 138

Designer enables you to create, maintain, and manage custom integration packages for use by the webMethods Integration Server. Often, many enterprise organizations employ a version control system (VCS) for the development of software solutions, providing automatic auditing, versioning, and security to software development projects. Such products include Microsoft Visual SourceSafe and IBM Rational ClearCase.

With the VCS Integration feature installed in your development environment, you can check packages or elements in to and out of your version control system (VCS). For example, to modify a flow service element, you would:

1. Check out the flow service. This automatically checks out its supporting files (node.ndf and flow.xml).
2. Modify the flow service in Designer and save the changes.
3. Check in the flow service element. This also checks in the node.ndf and flow.xml files and makes the files read-only when they are checked in.

Alternatively, if you want to work on other elements in addition to the flow service, you can check out the entire package.

For information about configuring VCS to work with Integration Server, see *Configuring the VCS Integration Feature*.

Note:
The VCS Integration feature provides functionality similar to that of local service development. However, the VCS Integration feature and local service development are not the same. For information about how the VCS Integration feature compares to local service development, see "How Does Local Service Development Differ from the VCS Integration Feature?" on page 101.

Important:
The WmVCS package, which provides the functionality for using the VCS Integration Feature, is deprecated as of Integration Server version 9.9. Software AG recommends that you use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer.

VCS Integration Supported Features

The use of a shared VCS repository greatly enhances team development of software solutions. The webMethods Version Control System Integration (VCS Integration) feature enables you to interact directly with a resident VCS repository in the following ways:

- Check in
- Check out
- Revert changes
- Delete
- Get latest version

- Get earlier version
- View history

VCS Integration Unsupported Features

The VCS Integration feature is intended for the control of webMethods packages and their contents in development environments only. The feature does not support:

- The ability to "diff" or merge files.
- The ability to move checked out elements loaded from an earlier VCS version.
- Static viewing of previous versions.
- VCS integration of files outside the package namespace structure (that is, components that do not appear in the Package Navigator view of Designer, or that exist outside the ..\packages\ns or the ..\packages\code\source directories).
- VCS integration of backup files generated by Designer or Integration Server.
- The ability to resolve conflicting files automatically. File conflicts between the VCS repository and your local file system must be resolved manually.

Locking Locally vs VCS Locking

In a shared development environment, there is typically a mechanism for a developer to lock a file during modification, and to unlock it when the modifications are complete. This prevents other developers from working on the file at the same time.

Designer and Integration Server provide basic locking and unlocking of project files on the Integration Server only, with no built-in interaction with an external VCS. When locking and unlocking is used, Integration Server files must be checked in to and out of the VCS manually, outside of the Integration Server or Designer.

The VCS Integration feature extends and is fully compatible with the basic locking functionality of Integration Server and Designer. When the VCS Integration feature is not installed or is disabled on an Integration Server, only the basic locking and unlocking functionality will be available in Designer.

System Locking and VCS Integration Feature

Designer and Integration Server support the concept of *system locking*. When an element's server-side files are marked read-only on the Integration Server, the element is *system locked*. Files are generally system locked when a server administrator accesses the file through the server's operating system and marks the files as read-only. You cannot edit an element until the server administrator makes the element's server-side files writable and you reload the package in which the element resides.

When an Integration Server has the VCS Integration feature enabled, system locking is effectively disabled for elements that are checked into the version control system with the VCS Integration

feature. The VCS Integration feature will override any read/write status changes applied manually by a server administrator.

About Unlocking Elements with Integration Server Administrator

The Integration Server Administrator enables you to unlock server files from the **Packages > Management > Locked Elements** page. However, because of architectural considerations, this unlocking mechanism is not tightly integrated with the VCS Integration feature.

As a result, Software AG strongly recommends that you do not use Integration Server Administrator to unlock elements that are managed with the VCS Integration feature. Doing so generally causes the element to enter an ambiguous state within Designer; the lock status may not be correct, and some VCS Integration feature menu commands may not be available.

If these conditions occur, refresh the Package Navigator view of Designer. If the condition persists, apply the **Check In** command. If you are still having problems, check in the element with the VCS client and restart the Integration Server.

Adding New Packages and Elements to a VCS

With the VCS Integration feature enabled, any new packages (or the supported elements within them) that you create in Designer are automatically added to the VCS and marked as checked out.

New packages and elements are added to the VCS regardless of how the package or element was created; it makes no difference if the package or element is created manually by the user, copied from another package or element, moved, renamed, or created from an adapter notification or a publishable document type. For information on the specific files that are subject to the VCS Integration feature, see *Configuring the VCS Integration Feature*.

Note:
The VCS Integration feature automatically adds C/C++ services to the VCS when they are created, but thereafter, you must check in and check out the supporting files for C/C++ services manually using the VCS client.

Adding Existing Packages and Elements to a VCS

Packages and the supported elements within them that were created prior to the installation of the VCS Integration feature can be added by applying the **Check In** or **Check Out** command, subject to the following behavior:

- When either command is applied to a package or a folder, all of the supported elements within the container's hierarchy are added to the VCS.
- When either command is applied to a folder or an element in a folder, the package and all of the folders in the path to the folder or element will also be added to the VCS. However, the remaining contents of the package and those higher-level folders will not be added to the VCS.
- When a folder containing coded services (Java and C/C++ for example) is added to the VCS, all of coded services in that folder will be added to the VCS.

- When you apply the **Check Out** command, the selected item is added to the VCS and then placed in a checked out state.

Be sure to check in the elements when you have finished working with them. For more information, see "Checking In Packages and Elements" on page 127. For more information on which element types are subject to the VCS Integration feature, see *Configuring the VCS Integration Feature*.

Modifying Elements that are in the VCS

When you modify a webMethods solution that you have checked out from the VCS repository, keep the following points in mind:

- Make sure to check in any elements that you modified with programs other than Designer (for example, HTML pages and DSPs).
- Make sure to check in any elements (flow services, documents, or other elements in the Package Navigator view) that you modified with Designer.

If you have previously worked with an earlier version of Designer, you may be experienced in checking individual files in and out of your VCS (such as flow.xml and source code files). However, the VCS Integration feature works at the package, folder, and element level. Individual files are checked into and out of your VCS automatically when you work with the package, folder, or element that contains them.

If you check out …	Integration Server checks out these files for you…
A package or a folder	■ All of the folders and elements within the package or folder, along with their supporting files.
A service	■ The node.ndf file for that service ■ Other supporting files, depending on the service type, such as flow.xml and java.frag files

Important:
The VCS Integration feature requires that you disable the **Automatically Unlock Upon Save** option in Designer, if you have implemented it. For information on disabling this option, see "Automatically Unlocking an Element Upon Saving" on page 93.

Checking Out Packages and Elements

Keep the following points in mind when checking out packages and elements from the VCS.

- The VCS Integration feature does not work with all files in the Integration Server directory structure. When you check out a package or element from the VCS repository, you must use the VCS client to manually check out any of its associated unsupported files that you want to work with, such as:

  IntegrationServer_directory\instances*instance_name*\config*.cnf

- If your package contains C/C++ services, you must check out those services manually with your VCS client and rebuild those services (for example, recompile the program to generate the DLL).
- Although packages and folders are never shown as checked out in the Package Navigator view, you can apply the **Check Out** command to a package or folder to check out all of the elements within the package or folder.
- When you check out a package or element from ClearCase, the check out is performed on the branch of the package or element that is currently present in the underlying view.
- When you check out an individual Java service, all coded services in the folder containing the Java service are checked out. If, for example, the folder contains three Java services and a C service, all four services are checked out when you check out any of the three Java services. When you check in any of the Java services in a folder, all coded services in the folder are checked in.
- If a package or element does not exist in the VCS, applying the **Check Out** command adds the package or element to the VCS repository. For more information about this behavior, see "Adding New Packages and Elements to a VCS" on page 125.
- Although the **Check Out** command can be applied at the package and folder level, the check mark is not applied to package and folder icons, only to the checked out elements within the package or folder.

To check out a package, folder, or element

- In Package Navigator view, right-click the package, folder, or element you want to work with and select **Check Out**.

  If a package or folder contains elements that are not supported by the VCS Integration feature (such as an adapter connection), you will receive a message naming the elements that will not be checked out. Checking out a package with many elements might take a significant amount of time. Designer will not be available during the check out procedure.

  After the check out procedure is completed, a small check mark appears to the right of each checked out element's icon in the Package Navigator view.

Checking In Packages and Elements

The VCS Integration feature adds packages and elements to your VCS when you apply the **Check In** or **Check Out** command to a file, a folder, or a package.

Keep the following points in mind when checking packages, folders, and elements into a VCS:

- Because the VCS Integration feature does not work with all files in the Integration Server directory structure, use the VCS client to manually check in any of the unsupported files that it uses (such as dynamic server pages, HTML documents, templates, and configuration files). For more information, see *Configuring the VCS Integration Feature*.

- Although packages and folders are never shown as checked out in the Package Navigator view, you can apply the **Check In** command to a package or folder to check in all of the elements within the package or folder.
- If your package contains C/C++ services, you must check in those services manually with your VCS client.
- When you check in a package or element to ClearCase, the check in is performed on the branch of the package or element that is currently present in the underlying view.
- If a package or element does not exist in the VCS, applying the **Check In** command adds the package or element to the VCS repository. For more information about this behavior, see "Adding New Packages and Elements to a VCS" on page 125.
- If a package or element has unsaved changes applied to it, you must save the changes to the package or element before you can check it in.

To check in a package, folder, or element

1. In Package Navigator view, right-click the package, folder, or element that you want to check in, and select **Check In**.
2. In the **Comment for Check In** dialog box, enter a comment that describes the changes made to the package or element. This comment will be displayed in the revision history for the element. If you do not enter a comment, the revision history will display only user, time, and date information for this check in.
3. Click **OK**. The small check mark is removed from the right of the package or element's icon in the Package Navigator view.

Notes:

- If the version of the file you checked out is not the same as the one that is currently available, another user has checked in the file in between. Your check in will not be done and an error message will be displayed.
- If you check in a package that existed before VCS Integration, the manifest.v3 file will not get checked into the VCS.
- If you have made no changes to the package, folder, or element, the comment you enter will be ignored by Visual SourceSafe unless you configure Visual SourceSafe to check in unchanged files.
- Checking in a package with many elements might take a significant amount of time. Designer will not be available during the check in procedure.

Reverting Changes to a Checked Out Package or Element

There may be times when you want to discard the changes made to a checked out package or element and return the package or element to its original version. This is commonly done when testing potential changes to files in the VCS repository.

The **Revert Changes** command discards all changes made to a package or element and replaces the Integration Server instance of the package or element with the most recent version checked into the VCS repository. This provides you with a sandbox feature for testing proposed changes. Only packages or elements that are checked out can be reverted.

Keep the following points in mind before reverting changes to a checked out package, folder, or element:

- Access Control List (ACL) permission settings are not revertible. If you make changes to ACL permissions for an element, and then revert the element, the ACL permissions will remain unchanged.
- You can use the **Revert Changes** command to reverse a check out command that you applied by mistake, even if no changes are made to the file.
- Although folders are never shown as checked out in the Package Navigator view, you can apply the **Revert Changes** command to a folder to revert changes to all of the elements in the folder's hierarchy.
- Reverting changes to a newly created element (prior to initial check in) may cause the element to be corrupted. Software AG recommends that you check in a new element immediately after creation and then check it out again before you enter any modifications.
- When you apply the **Revert Changes** command to a package, or to a folder in a package, all of the supported elements within the container's hierarchy that are currently checked out are reverted.
- The **Revert Changes** command reloads the entire package containing the element; this may cause sessions currently using services in the package to fail.

To revert changes to a checked out package, folder, or element

1. In Package Navigator view, right-click the package, folder, or element for which you want to revert changes, and select **VCS > Revert Changes**.

2. In the Revert Node alert window, click **OK** to confirm reverting the element to the most recent version in the VCS repository.

   The small check mark is removed from the right of the element's icon in the Package Navigator view, the checked out files are replaced with the most recent version checked in to the VCS repository, and the entire package is reloaded.

   Note:

When you work with a package with many elements, it may take a significant amount of time to check the package in or out. Designer will not be available during the check in or check out procedure.

Getting the Latest Version from the VCS

The VCS Integration feature provides the **Get Latest Version** command to replace packages, folders, and elements on Integration Server with the most recent version in the VCS repository (referred to as a sync operation in some VCS programs).

Keep the following points in mind when retrieving the latest version of a package, folder, or element from a VCS repository.

- You cannot apply the **Get Latest Version** command to checked out packages, folders, or elements. You cannot apply the command to a package or a folder if any of the supported elements within the container's hierarchy are checked out.
- Although packages and folders are never shown as checked out in the Package Navigator view, you can apply the **Get Latest Version** command to a package or folder to get the latest version of all of the elements within the package or folder.
- For ClearCase, the **Get Latest Version** command updates the package, folder, or element in the branch configured for the view. Do not use the **Get Latest Version** command for Dynamic Views because Dynamic Views always contain the latest version.
- Some folders or elements might be deleted after you apply the **Get Latest Version** command. This will occur when there is no current version of that folder or element (that is, it has been deleted from the VCS repository since the last update).
- The **Get Latest Version** command reloads the entire package containing the element. This might cause sessions currently using services in the package to fail.

To get the latest version of package, folder, or element

- In Package Navigator view, right-click the package, folder, or element for which you want to retrieve the latest version, and select **VCS > Get Latest Version**.

Note:
If a folder contains locked non-Java elements and you apply the **Get Latest Version** command to an unlocked Java element in that folder, the error message "Subnode(s) checked out" displays. Check in all elements in the folder and try again.

Getting an Earlier Version from the VCS

The VCS Integration feature provides the **Get Earlier Version** command to replace packages, folders, and elements on the Integration Server with an earlier version in the VCS repository.

Keep the following points in mind when retrieving an earlier version of a package, folder, or element:

- To retrieve an earlier version, you must know the date or VCS label of the earlier version. You can obtain this information with the **View History** command. (For more information, see "Viewing the History of a Folder or Element" on page 135.)

  The VCS Integration feature does not provide the ability to assign VCS labels; you must use your VCS client to assign a label if one does not exist. To eliminate any problems, Software AG recommends that you apply the version label at the package level, thereby including all folders and elements within the package hierarchy.

- You cannot apply the **Get Earlier Version** command to checked out packages, folders, or elements.

- You cannot apply the command to a package or a folder if any of the supported elements within the container's hierarchy are checked out.

- Most VCS servers will not permit you to copy, move, delete, or rename earlier versions of elements. Software AG recommends that you do not apply any of these actions to an earlier version file, as unpredictable results might occur.

- Although packages and folders are never shown as checked out in the Package Navigator view, you can apply the **Get Earlier Version** command to a package or folder to get an earlier version of all of the elements within the package or folder.

- For ClearCase, the **Get Earlier Version** command loads the version of the branch indicated by the **ClearCase Branch Name** field in Integration Server Administrator (see *Configuring the VCS Integration Feature*). If no branch is identified in that field, ClearCase loads the main ClearCase branch. Do not use the **Get Earlier Version** command for Dynamic Views because Dynamic Views always contain the latest version.

- Some folders or elements might be deleted after you apply the **Get Earlier Version** command. This will occur when there is no earlier version of that folder or element (that is, it has been added to the VCS repository since the creation of the version being retrieved).

- When you apply the **Get Earlier Version** command to a Java service, the earlier version will be loaded for all Java services in that folder, as well as for all folders and elements contained in the folder.

- The **Get Earlier Version** command also reloads the entire package containing the element. This might cause sessions currently using services in the package to fail.

To get an earlier version of a package, folder, or element

1. In Package Navigator view, right-click the package, folder, or element for which you want to retrieve the latest version, and select **VCS > Get Earlier Version**.

2. In the Get Earlier Version dialog box, do one of the following:

Select...	To...
Date	Retrieve an earlier version using the date (Visual SourceSafe).

Select...	To...

	In the field next to **Date**, enter the VCS repository date and time of the version you want to retrieve.
	
	You must type the date and time using the Designer format MM/DD/YY HH:MM:SS, where HH:MM:SS is in 24-hour time format. For example, in the History dialog box, the VCS time is presented in the format:
	
	`User: UserName  Date: 1/13/06  Time: 2:56p`
	
	This signifies January 13, 2006, 14:56 hours. You must type the date into the **Date** field in this format:
	
	`01/13/06 14:56`
	
	Do not include the time zone (for example, EST) when typing the date and time. The precision of the specified time (that is, whether minutes and seconds are accepted, or minutes only) is determined by the time format of the VCS application. For example, Visual SourceSafe dates files with minutes only.

Label	Get an earlier version by providing the VCS label (Visual SourceSafe and ClearCase) or to get an earlier version by providing a version number (ClearCase). In the field next to **Label**, enter the VCS label text or version number.

3. Click **OK**. All supported and checked in elements are updated to the specified version in the VCS repository.

Notes:

- If a folder contains locked elements that are not Java services and you apply the **Get Earlier Version** command to an unlocked Java service in that folder, the error message "Subindex() checked out" displays. Check in all elements in the folder and try again.
- If you edit an element that you retrieved using the **Get Earlier Version** command, the VCS Integration feature will not view it as the most current version and therefore will not allow you to check it in. When you attempt to check in the element, an "out of date" error message displays. Update the element to the latest version before applying your changes.

Deleting Packages and Elements from the VCS

The VCS Integration feature enhances the existing **Delete** command to delete packages, folders, and elements from both Integration Server and the VCS repository.

Keep the following points in mind when deleting items from an Integration Server on which the VCS Integration feature is configured:

- The **Delete** command can be applied to packages, folders, and elements regardless of the checked in/checked out status of the elements.

- You cannot delete packages or elements checked out by other users.
- When the **Delete** command is applied to a package, Integration Server saves a copy of the package in the package recovery area of Integration Server.
- When the **Delete** command is applied to a folder or element in a folder (a service, for example), the folder or element is deleted completely from the Integration Server. For those VCS servers that support restorable deletion, the element is deleted from the repository view but not destroyed.
- When you apply the **Delete** command to a package, or to a folder in a package, all of the elements within the container's hierarchy (both VCS elements and non-VCS elements) are deleted.
- For ClearCase, the **Delete** command removes the package, folder, or element from the versioned object base (VOB), so that the package, folder, or element is removed from all branches.
- Visual SourceSafe and ClearCase do not support entry of a delete comment. The VCS history will show only the VCS user name and the time of deletion.

Note:
Do not apply the **Delete** command to any earlier version packages or elements that are checked out, as unpredictable results may occur.

To delete a package, folder, or element from both Integration Server and the VCS

1. In Package Navigator view, select the package, folder, or element you want to delete.
2. Select **Edit > Delete**.

   The Delete Confirmation dialog box appears. If you are deleting a publishable document type, Designer prompts you to delete the associated Broker document type as well. For more information about deleting Broker document types, see "Deleting Elements" on page 61.
3. Click **OK**.

   If a VCS package or element is checked out by another user, the package or element will not be deleted, and an error message appears. In addition, any parent folders leading to the checked out element will not be deleted.

   When deleting a package, a message box appears stating that the deletion is complete and that the deleted package has been copied to the recovery area of the Integration Server. If this message appears, click **OK**.

Restoring Deleted Items

After you delete a package, folder, or element from both Integration Server and the VCS repository, you may want to restore the package, folder, or element back into the Integration Server from the VCS server.

Restoring a Deleted Package

To restore a deleted package

1. Use Integration Server Administrator to recover the deleted package.

   Note:
   Packages are best restored with the Recover Packages feature of the Integration Server Administrator (only administrator users can recover a package)

2. In the Service Development perspective, refresh the Package Navigator view to display the recovered package.

3. Right-click the package and select **Check In**.

Restoring a Deleted Folder or Element

You can restore a deleted folder or element (a service, for example) in the VCS repository if you did not subsequently delete it completely from the VCS repository.

To restore a folder or element that has been deleted from both Integration Server and the VCS

1. With your VCS client, restore the folder or element in the VCS repository.

   Important:
   If the folder or element contains subfolders or individual files, be sure you restore all of the folder or element contents, typically by applying a *recursive* restore. If you restore a service, for example, without restoring its node.ndf file, the service will not be recognized by Integration Server.

2. Using the VCS client, check out the restored element to its original location in the ..\packages\ns directory. You may receive a message that the folder or element does not exist, with a request to create the folder or element. If so, respond so that all folders, elements, and files are created.

3. In Package Navigator, right-click the package that contained the deleted element and select **Reload Package**. This displays the restored element in the Package Navigator view.

   Note:
   At this point, although the restored element is in a checked out state on the VCS server, it does not display the checked out symbol in the Package Navigator view.

4. In Package Navigator view, right-click the restored element and select **Check In**.

Copying and Moving Folders or Elements

The VCS Integration feature integrates with the copy, paste, and move functionality of Designer. The results of all these actions are applied to the VCS repository. You cannot copy and paste, move an earlier version of a package, folder, or element.

Important:
When you move or rename a folder or element, you are effectively creating a new entity in the VCS repository. Therefore, the previous folder or element is deleted and a new entity is created. This means that a new revision history is started for the moved or renamed entity as well. If you want to view previous revision information for a moved or renamed folder, or element, you must locate the deleted version of the file in the VCS repository and view the revision history there.

Because the default behavior of the VCS Integration feature is to add any new folder or element to the VCS repository, a copied or moved folder or element automatically appears in the VCS repository in its new location. In the case of a moved item, the previous location is deleted from both Designer and the VCS repository.

A copied folder or element is always placed in a checked out state in its new location. A moved folder or element retains its checked in or checked out state; special conditions apply when you copy or move a coded service. For more information on copying and moving coded services, see "Working with Java Services" on page 138.

When you move an element that has dependents, the dependents will not be updated until you check in the moved element. This may cause failure of any services that use the dependent elements.

Renaming Packages, Folders, and Elements

The VCS Integration feature integrates with the rename functionality of Designer. The results of all these actions are applied to the VCS repository. You cannot rename an earlier version of a package, folder, or element.

When you rename a package, folder, or element, the renamed item retains its checked in or checked out state, and the new name is applied in the VCS repository. Also, note that when you rename an element that has dependents, the dependents will not be updated until you check in the renamed element. This may cause failure of any services that use the dependent elements.

Viewing the History of a Folder or Element

The VCS Integration feature provides the **View History** command to display the revision history of a selected folder or element. The **View History** command is not available at the package level.

- If the folder or element exists in the VCS repository, the lock status and VCS revision history is displayed.
- If the folder or element is not contained in the VCS repository (that is, it exists only on Integration Server), only the lock status of the element is displayed.

- If a selected element contains more than one file (for example, a flow.xml file and node.ndf file in a flow service), the VCS revision history presents information for each file.
- For ClearCase, the **View History** command displays the history of the folder or element in all branches that contain the file.

Note:
Technically, folders are not checked in or checked out of the VCS repository; it is actually the elements within the folder that are checked in or out. When viewing the history for a folder, you are actually viewing the history of the node.idf file within the folder.

When you move or rename a folder or element, the previous folder or element is deleted and a new entity is created. This means that a new revision history is started for the moved or renamed entity as well. If you want to view previous revision information for a moved or renamed folder, or element, you must locate the deleted version of the file in the VCS repository and view the revision history there.

To view the history of an element

1. In Package Navigator view, right-click the element for which you want to view history and select **View History**.

   The **elementName History** dialog box appears. If necessary, you can copy text from the dialog box and paste it into another program. For more information about version history details, see "Version History Details" on page 136.

2. When you have finished viewing the history information, click **OK**.

Version History Details

The following table identifies the available version history information for files, supporting folders, and elements in the VCS repository.

Information	Definition
User	The VCS user account name under which the revision was executed.
Date	The date applied to the revision by the VCS server.
Time	The time applied to the revision by the VCS server.
Checked In	The full VCS project path for the checked in file.
Comment	Text entered by the user at check in time. This may contain no text if the user did not enter a comment.
dev_user	The Designer user under which the revision was executed.
is_host	The name of the host on which Integration Server is running.

Information	Definition
dev_host	The name of the host on which Designer is running.
is_time	The date and time applied to the revision by Integration Server.
Label	The VCS label applied to the file. If no VCS label exists, this entry is not displayed.
Label Comment	Text entered by the user when the label was applied. This may contain no text if the user did not enter a comment.

Working with Blaze Rules Services

When you deploy a rule from Blaze Advisor, a new package, a folder, a subfolder and a rule node are created in Integration Server. However, only the package is checked in to the VCS repository, during the rule deployment. You must check in or check out the folders, subfolders, and rule node manually.

Working with Web Service Descriptors

When a web service descriptor is created in Designer, a web service descriptor node and associated schemas and document types are created in Integration Server. However, only the web service descriptor node is checked in to the VCS repository. You must check in or check out the associated schemas and document types manually.

Working with webMethods Adapter Connections

After they are created, adapter connections appear within a designated package in the Package Navigator view of Designer. Adapter connections are the one exception to the rule that all elements shown in the Package Navigator view of Designer are subject to VCS commands.

Because they are created and maintained in the Integration Server Administrator, adapter connections cannot be checked in or out of the VCS repository from Designer. You must create, configure, and modify adapter connections using the Integration Server Administrator (you must have webMethods administrator privileges to access the adapter administration screens).

If you want to maintain your adapter connection files in the VCS repository, you must do so manually with your VCS client.

Note:
Because the adapter connection appears within a package, there will be a corresponding folder created within the VCS repository. However, this folder contains only the *.ndf file that defines the folder; no adapter connection files are placed there.

Also note that a publishable document type for an adapter notification cannot be directly checked in to and out of the VCS repository. It is automatically checked in or out when you use the VCS client to manually check in or check out its associated adapter notification.

Working with Java Services

When you copy or move a Java service that is registered in a VCS repository, the service's checked in or checked out status in the destination folder is affected according to the action taken. In addition, certain restrictions apply to the labeling of Java services.

Copying Java Services

When you copy a Java service, the copied service is added to the VCS repository and placed in a checked out state in its new folder location, regardless of the check in/check out status of the source Java service. All other Java services in the destination folder will be checked out if they are not already checked out.

You cannot copy or move an earlier version of a coded service retrieved from the VCS.

Moving Java Services

The ability to move a Java service is available only with the latest version of a Java service. If you load an earlier version of a Java service, it cannot be moved.

When you move a Java service, the results are influenced by the state of the Java service before it is moved (that is, checked in or checked out), and the contents and state of the destination folder.

- When a checked out Java service is moved, it will always be checked out in the destination folder. If there are other coded services in the destination folder they too will be checked out (if they are not already checked out).
- When a checked in Java service is moved to a destination folder that contains no other checked out coded services, its checked in state will be maintained.
- When a checked in Java service is moved to a destination folder that contains one or more checked out coded services, its state will change to checked out.

Note:
When you move a Java service, its VCS revision history is reset. To retrieve the earlier information, you must find the previous version of the file as a deleted item in its previous VCS location and view the revision history there.

Labeling Java Services in the VCS

The VCS Integration feature enables you to get an earlier version of VCS files by specifying a label, as described in "Getting the Latest Version from the VCS" on page 130. VCS labels must be created with the VCS client; the VCS Integration feature does not enable you to create VCS labels from within Designer.

The files supporting a Java service are stored in two locations within the package directory structure in the VCS repository:

- ..*package*\code\source

- ..*package*\ns*folderName**JavaServiceNameFolder*

The files in these directories must have the same label for the entire Java service to be retrieved by label.

To minimize any problems, Software AG recommends that you apply the version label at the package level, thereby including all folders and elements within the package hierarchy.

8 Managing Packages

- Creating a Package 142
- Documenting a Package 143
- Viewing Package Settings, Version Number, and Patch History 144
- Assigning a Version Number to a Package 145
- About Copying Packages Between Servers 145
- Reloading a Package 147
- Comparing Packages 148
- Deleting a Package 148
- Exporting a Package 148
- About Package Dependencies 149
- Assigning Startup, Shutdown, and Replication Services to a Package 151

A package is a container that is used to bundle services and related elements, such as specifications, IS document types, IS schemas, and triggers. When you create a folder, service, IS document type, or any element, you save it in a package.

Packages are designed to hold all of the components of a logical unit in an integration solution. For example, you might group all the services and files specific to a particular marketplace in a single package. By grouping these components into a single package, you can easily manipulate them as a unit. For example, you can copy, reload, distribute, or delete the set of components (the "package") with a single action.

Although you can group services using any package structure that suits your purpose, most sites organize their packages by function or application. For example, they might put all purchasing-related services in a package called "PurchaseOrderMgt" and all time-reporting services into a package called "TimeCards."

On the server, a package represents a subdirectory within the *IntegrationServer_directory*\instances*instance_name*\packages directory. All the components that belong to a package reside in the package's subdirectory.

Creating a Package

When you want to create a new grouping for services and related files, create a package. When you create a package, Designer creates a new subdirectory for the package in the file system on the machine where the Integration Server is installed. For information about the subdirectory and its contents, see *webMethods Integration Server Administrator's Guide*.

To create a package

1. In Designer: **File > New > Package**

2. In New Integration Server Package dialog box, select the Integration Server on which you want to create the package.

3. In the **Name** field, type the name for the new package using any combination of letters, numbers, and the underscore character. For more information, see "Guidelines for Naming Packages" on page 142.

4. Click **Finish**. Designer refreshes the Package Navigator view and displays the new package.

Guidelines for Naming Packages

Keep the following guidelines in mind when naming new packages:

- Start all package names with an uppercase letter and capitalize the first letter of subsequent words (for example, PurchaseOrder).
- Keep package names short. Use abbreviations instead of full names. For example, instead of ProcessPurchaseOrder, use ProcessPO.

- Make sure the package name describes the functionality and purpose of the services it contains.
- Avoid creating package names with random capitalization (for example, cOOLPkgTest).
- Avoid using articles (for example, “a,” “an,” and “the”) in the package name. For example, instead of TestTheService, use TestService.
- Do not use the prefix “Wm” in any case combination. Integration Server and Designer use the “Wm” prefix for predefined packages that contain services, IS document types, and other files. Additionally, custom packages with a “Wm” prefix can be problematic when deploying the packages using Deployer.
- Avoid using control characters and special characters like periods (.) in a package name. The watt.server.illegalNSChars setting in the server.cnf file (which is located in the *IntegrationServer_directory*\instances*instance_name*\config directory) defines all the characters that you cannot use when naming packages. Additionally, the operating system on which you run the Integration Server might have specific requirements that limit package names.

Documenting a Package

You can communicate the purpose and function of a package and its services to other developers by documenting the package.

To create documentation for a package

1. Document the package in one or more web documents (such as HTML pages). Be sure to name the home page for the package documentation `index.html`. The index.html file can contain links to the other web documents for the package. An index.html file exists for each package installed by the Integration Server.

2. Place the documents in the pub subdirectory for the package on the Integration Server.

   For example, place the package documentation for a package named “PurchaseOrders” in the following directory:
   IntegrationServer_directory\instances*instance_name*\packages\PurchaseOrders\pub

   Tip:
   An alternate location for package documentation is the *IntegrationServer_directory*\instances*instance_name*\packages\doc directory. Typically, this directory is used for reference material such as PDFs that do not need to be published to the web.

Accessing Package Documentation

To access documentation for a package

- Enter the URL for the package documentation. The URLs for package documentation have the following format:

http://*serverName*:*port*/*PackageName*/*DocumentName*

where:

serverName:*port*	is the name and port address of Integration Server on which the package resides.
PackageName	is the name of the package for which you want documentation.
DocumentName	is the name of the web document you want to access. If you do not specify a *DocumentName*, Integration Server automatically displays the index.html file.

Viewing Package Settings, Version Number, and Patch History

For each package, Designer tracks and displays settings, version number, and a history of installed patches. A patch is a partial upgrade, change, or fix to the contents of a package.

You might want to check the settings or patch history of a package for the following reasons:

- To determine which version of the package is installed.
- To avoid overwriting the installed package with a lower version of the same package.
- To view the changes that are included in each version of the package.
- To inform Software AG Global Support which versions of predefined packages are installed on your Integration Server.

To view package settings, version number, and patch history

1. In Package Navigator view, select the package whose properties you wish to view.
2. Click **File > Properties**.
3. In Properties for `PackageName` dialog box, select **Package Settings**.

   The **Package Settings** page displays the version and patch history for the package since the last full release of the package. (A full release of a package incorporates all previous patches for the package.) For more information about package settings, see "Package Properties" on page 957.

Note:
When the server administrator installs a full release of a package (a release that includes all previous patches for the package), the Integration Server removes the existing patch history. This helps the server administrator avoid potential confusion about version numbers and re-establish a baseline for package version numbers.

Assigning a Version Number to a Package

You can assign a version number to a package to identify different versions of the package. For example, you might want to assign a new version number to a package when you add new services to the package or after you fix bugs in a package. You might find assigning version numbers especially helpful if you work in a development environment where more than one person makes changes to a package.

Keep the following in mind when assigning version numbers to packages:

- By default, Designer assigns the version number 1.0 to each package that you create.
- When you change the version number of a package, make sure that you update the package dependencies for other packages that depend on the earlier version of this package.
- Assign and change package version numbers through Designer only when the packages are in a development stage. To avoid difficulties installing package releases, do not change version numbers on packages you receive from trading partners, packages to which you subscribe, or packages installed with Integration Server.

To assign a version number to a package

1. In Package Navigator view, select the package to which you want to assign a version number.
2. Click **File > Properties**.
3. In Properties for *PackageName* dialog box, select **Package Settings**.
4. In the **Package Version** field, type the version number you want to assign to the package. Be sure to format the version number in one of the following ways: *X.X* or *X.X.X* (for example, `1.0`, `2.1`, `2.1.3`, or `3.1.2`).
5. Click **OK**.

About Copying Packages Between Servers

You can copy a package to another Integration Server in one of two ways:

- From Designer.

  You can copy a package and its contents to another Integration Server from within Designer by performing a copy or a drag-and-drop action. Copying packages using either of these methods provides a quick way to share a set of services and their supporting files with other developers in a development environment.

- From Integration Server Administrator.

  You can also copy a package from within the Integration Server Administrator by replicating the package. You can then send, or publish, the package to other Integration Servers. Copying

packages using this method allows you to customize the way in which packages are replicated and published. This method is useful for managing releases between development and production environments, for deploying releases to partners or customers, or for distributing package updates to developers working in large, collaborative environments.

For information about replicating packages and managing releases from within Integration Server Administrator, see *webMethods Integration Server Administrator's Guide*.

Copying Packages

When copying packages, keep the following points in mind:

- You can copy a package to a different server only if you are a member of a group assigned to the Replicators ACL on the source and destination servers and you are logged on to both servers.
- Before you copy a package that contains elements with unsaved changes, you must save the changes.
- You cannot undo a copy action using the **Edit > Undo** command.
- If you copy a package that depends on other packages to load (that is, the package has package dependencies), and the required packages are not present on the destination server, the package will be copied but it will not be enabled.
- You cannot copy a package to another server if the destination server already contains a package with that name.

  Note:
  Because UNIX directories are case sensitive, Integration Servers running in a UNIX environment will allow packages with similar names to reside on the same server. For example, you can copy a package named orderProcessing to a server that contains a package named OrderProcessing.

- When you copy a package from another Integration Server, it is possible that an HTTP URL alias associated with the new package has the same name as an HTTP URL alias already defined on your Integration Server. If Integration Server detects a duplicate alias name, it will write a message to the server.log.
- When you copy a package from another Integration Server, it is possible that a port alias for a port in the new package has the same alias as a port already defined on your Integration Server. A port alias must be unique across the Integration Server. If Integration Server detects a duplicate port alias, it will not create the port and will write the following warning to the server.log:

  [ISS.0070.0030W] Duplicate alias *duplicateAliasName* encountered creating *protocol* listener on port *portNumber*

  Note:
  If you want the port to be created when the package is loaded, use Integration Server Administrator to delete the existing port with that alias, create a new port that has the same

properties as the just deleted port, and then reload the package containing the port with the duplicate alias. Integration Server creates the port when the package is reloaded.

- When you copy a package from a version of Integration Server prior to version 9.5 SP1 to an Integration Server version 9.5 SP1, Integration Server creates an alias for each port associated with the package. Integration Server assigns each port an alias. For information about the naming conventions used Integration Server, see the *webMethods Integration Server Administrator's Guide*.
- If the package you are copying is associated with an e-mail listener, Integration Server will install the package but will not enable the listener. This is because the password required for the Integration Server to connect to the e-mail server was not sent with other configuration information about the listener. To enable the listener, go to the **Security > Ports > Edit E-mail Client Configuration** Screen and update the **Password** field to specify the password needed to connect to the e-mail server.

To copy a package between servers

1. In Package Navigator view, select the package you want to copy.
2. Click **Edit > Copy**.
3. If the package you want to copy contains elements with unsaved changes, Designer alerts you that the package cannot be copied until you save the changes. Click **OK** to close the alert dialog box. Then, save the changes and repeat the copy action.
4. Select the server where you want to copy the package.
5. Click **Edit > Paste**.

Reloading a Package

Sometimes, you need to reload a package on the server to activate changes that have been made to it outside of Designer. You need to reload a package if any of the following occurs:

- A Java service that was compiled using jcode is added to the package.
- New jar files are added to the package.
- Any of the configuration files for the package are modified.

Note:
Reloading a package is not the same as refreshing the Package Navigator view. When you refresh the Package Navigator view, Designer retrieves a fresh copy of the contents of all the packages from the memory of the Integration Server. When you reload a package, Integration Server removes the existing package information from memory and loads new versions of the package and its contents into its memory.

To reload a package

1. In Package Navigator view, select the package you want to reload.

2. Right-click the package and click **Reload Package**.

Comparing Packages

You can use the compare tool to compare two packages on the same server or on different servers. For more information, see "Comparing Integration Server Packages and Elements" on page 921.

Deleting a Package

When you no longer need the services and files in a package, you can delete the package. Deleting a package removes the package and all of its contents from the Package Navigator view.

When you delete a package from Designer, Integration Server saves a copy of the package. If you later want to recover the package and its contents, contact your server administrator. Only Integration Server Administrator users can recover a package. For more information about recovering packages, see *webMethods Integration Server Administrator's Guide*.

Before you delete a package, make sure that:

- Other users or other services do not use (depend on) the services, templates, IS document types, and schemas in the package. You can use the **Package Dependencies** option to identify other services that are dependent on a service in a package that you want to delete. For more information, see "Identifying Package Dependencies" on page 149.
- All elements in the package that you want to delete are unlocked, or locked by you. If the package contains elements that are locked by others or system locked, you cannot delete the package.

To delete a package

1. In Package Navigator view, select the package you want to delete.

2. Click **Edit > Delete**.

Exporting a Package

Packages can be exported to your hard drive so that they can be shared with partners or developers. You can install an exported package on another server by using the package publishing functionality in the Integration Server Administrator. Locking information is not exported.

To export a package

1. In Package Navigator view, select the package you want to export to your hard drive.

2. Right-click the package and click **Export from Server**.

3. In the Save As dialog box, select the location on your hard drive where you want the exported package to reside. Click **Save**.

   This exports the package to a ZIP file and saves it on your hard drive. The ZIP file can then be published on another server.

Note:
The **Export from Server** option is not the same as the **File > Export** option. With **File > Export**, you can export files from the Workbench to the file system.

About Package Dependencies

If a package needs elements from another package to load before it can load, you must set up package dependencies. For example, you must identify package dependencies if a startup service for a package invokes a service in another package. The startup service cannot execute if the package containing the invoked service has not yet loaded.

Additionally, you should set up a package dependency if a service uses a document type from a different package as the input or output signature.

You must also identify package dependencies if Java services in a package need to access Java classes contained in another package.

Important:
Other webMethods components might include packages that register new types of elements in Designer. You should save instances of these new element types in packages that list the registering package as a package dependency. The registering package needs to load before your packages so that Designer can recognize instances of the new element type. For example, if you create new flat file schemas, you must save the flat file schemas in packages that identify the WmFlatFile package as a package dependency.

Identifying Package Dependencies

Keep the following in mind when creating package dependencies:

- When you identify a package dependency, you must indicate the version number of the package that needs to load first. For example, the “Finance” package might depend on version 2.0 of the “FinanceUtil” package. It is possible that the services and elements needed by a dependent package are contained in more than one version of the same package. For example, the “Finance” package might depend on version 2.0 or later of the “FinanceUtil” package.
- Make sure that you do not create circular package dependencies. For example, if you identify “FinanceUtil” as a dependent package for the “Finance” package, do not identify “Finance” as a dependent package for the “FinanceUtil” package. If you create circular package dependencies, neither package will load the next time you start the Integration Server.

- If you create new adapter services and adapter notifications, you should save them in packages that identify the webMethods *AdapterName* package as a package dependency.
- Only one version of a package can be installed at one time. If the available version of the package specified in the package dependency is not the correct version, Integration Server does not load the dependent package. Integration Server writes a dependency load error for the dependent package to the server log.

To identify a package dependency

1. In Package Navigator view, select the package for which you want to specify package dependencies.
2. Click **File > Properties**.
3. In Properties for *PackageName* dialog box, select **Package Dependencies**.
4. Click .
5. In the Add Dependent Package dialog box, enter the following information:

In this field...	Enter...
Package	The name of the package you want Integration Server to load prior to loading the package selected in the Package Navigator view. Type the name of the package in the **Package** field or click **Browse** to choose the dependent package.
Version	The version number of the package you specified in the **Package** field. More than one version of the same package might contain the services and elements that a dependent package needs Integration Server to load first. A dependency declared on a version is satisfied by a package with a version that is equal to or greater than the specified version. For example, to specify versions 3.0 or later of a package, type `3.0` for the version number. To specify versions 3.1 or later, type `3.1.0` for the version number. You can also use an asterisk (*) as a wildcard in the version number to indicate that any version number equal to or greater than the specified version will satisfy the package dependency. If any version of the package satisfies the package dependency, type `*.*` as the version number.

6. Click **OK**.
7. Click **OK** in the Properties for *PackageName* dialog box.

Removing Package Dependencies

Use the following procedure to remove a package dependency that is no longer needed. For example, if you delete the service in "Finance" that invokes the service in "FinanceUtil," then you would no longer need a package dependency on the "FinanceUtil" package. Another case where you would remove the package dependency is if you move the services in the "FinanceUtil" package into the "Finance" package.

To remove a package dependency

1. In Package Navigator view, select the package for which you want to remove a package dependency.
2. Click **File > Properties**.
3. In Properties for *PackageName* dialog box, select **Package Dependencies**.
4. Select the package dependency you want to remove and click .
5. Click **Yes** to confirm the deletion.
6. Click **OK** in the Properties for *PackageName* dialog box.

Assigning Startup, Shutdown, and Replication Services to a Package

You can set up services to automatically execute each time Integration Server loads, unloads, or replicates a package. These types of services are called startup, shutdown, or replication services.

What Is a Startup Service?

A startup service is one that Integration Server automatically executes when it loads a package into memory. The server loads a package:

- At server initialization (if the package is enabled).
- When someone uses Designer or the Integration Server Administrator to reload a package.
- When someone uses Designer or the Integration Server Administrator to enable a package.

A startup service is one that Integration Server automatically executes when it loads a package into memory. The server loads a package:

Assigning a Startup Service

Keep the following guidelines in mind when assigning startup, shutdown, and replication services to packages:

- When you assign a startup or shutdown service to a package, you can only assign a service that resides in the same package. For example, a startup service for the "Finance" package must be located in the "Finance" package.
- Because services in a package are not made available to clients until the package's startup services finish executing, you should avoid implementing startup services that access busy remote servers. They will delay the availability of other services in that package.
- You can assign one or more startup services to a package; however, you cannot specify the order in which the services execute. If you have a series of startup services that need to execute in a specific order, create a "wrapper" service that invokes all the startup services in the correct order. Designate the "wrapper" service as the startup service for the package.
- If a startup service invokes a service in another package, make sure to identify the other package as a package dependency for the package containing the startup service.

To assign a startup service

1. In Package Navigator view, select the package to which you want to assign startup services.
2. Click **File > Properties**.
3. In Properties for *PackageName* dialog box, select **Startup/Shutdown Services**.
4. Under **Startup services**, select the service from the **Available Services** list, and click ➪.

   Repeat this step for each service you want to add as a startup service for the package.

   Note:
   A service that you just created does not appear in the **Available Services** list if you have not refreshed your session on the server since you created the service.
5. Click **OK**.

Removing a Startup Service

If you remove a startup service that invoked a service in another package and the package was identified as a package dependency, make sure you remove the package dependency after you remove the startup service.

To remove a startup service

1. In Package Navigator view, select the package for which you want to remove startup services.
2. Click **File > Properties**.

3. In Properties for *PackageName* dialog box, select **Startup/Shutdown Services**.

4. Under **Startup services**, select the service you want to remove from the **Selected services** list, and click ⇦.

5. Click **OK**.

What Is a Shut Down Service?

A shutdown service is one that the Integration Server automatically executes when it unloads a package from memory. The server unloads a package from memory:

- At server shutdown or restart.
- When someone uses the Integration Server Administrator to disable the package.
- When someone uses the Integration Server Administrator to reload a package before it is removed from memory.

Shutdown services are useful for executing clean-up tasks such as closing files and purging temporary data. You could also use them to capture work-in-progress or state information before a package unloads.

Assigning a Shutdown Service

When you assign a shutdown service to a package, you can only assign a service that resides in the same package. For example, a shutdown service for the "Finance" package must be located in the "Finance" package.

To assign a shutdown service

1. In the Package Navigator view, select the package to which you want to assign shutdown services.

2. Click **File > Properties**.

3. In the Properties for the *PackageName* dialog box, select **Startup/Shutdown Services**.

4. Under **Shutdown Services**, select the service from the **Available services** list, and click ⇨.

   Repeat this step for each service you want to add as a shutdown service for the package.

Removing a Shutdown Service

To remove a shutdown service

1. In the Package Navigator view, select the package for which you want to remove shutdown services.

2. Click **File > Properties**.

3. In the Properties for the *PackageName* dialog box, select **Startup/Shutdown Services**.

4. To remove a shutdown service, under **Shutdown services**, select the service you want to remove from the **Selected services** list, and click .

5. Click **OK**.

What Is a Replication Service?

A replication service is one that Integration Server automatically executes when it prepares to replicate a package. A replication service executes when the Integration Server Administrator creates a package release (full release or patch) or creates a package archive.

Replication services provide a way for a package to persist state or configuration information so that these are available when the published package is activated on the remote server.

Note:
The term replication service does not refer to the services contained in pub.replicator or to services that subscribe to replication events (replication event services).

Assigning a Replication Service

When you assign a replication service to a package, you can assign any service from any loaded package on Integration Server, including the current package.

To assign a replication service

1. In the Package Navigator view, select the package to which you want to assign replication services.

2. Click **File > Properties**.

3. In Properties for *PackageName* dialog box, select **Replication Services**.

4. Click .

5. In the Select a replication service dialog box, select the service that you want to use as a replication service.

6. Click **OK**.

Repeat these steps for each service you want to add as a replication service.

Removing a Replication Service

To remove a replication service

1. In the Package Navigator view, select the package for which you want to remove replication services.
2. Click **File > Properties**.
3. In Properties for *PackageName* dialog box, select **Replication Services**.
4. Select the replication service you want to remove and click .
5. Click **Yes** to confirm the deletion.
6. Click **OK**.

9 Building Services

- A Process Overview 158
- Package and Folder Requirements 159
- About the Service Signature 159
- About Service Run-Time Parameters 165
- About Automatic Service Retry 178
- About Service Auditing 180
- Using a Circuit Breaker with a Service 187
- About Universal Names for Services or Document Types 191
- About Service Output Templates 195
- Printing a Flow Service 197
- Comparing Flow Services 198

Services are method-like units of logic that operate on documents. They are executed by Integration Server. You build services to carry out work such as extracting data from documents, interacting with back-end resources (for example, submitting a query to a database or executing a transaction on a mainframe computer), and publishing documents to the Broker. Integration Server is installed with an extensive library of built-in services for performing common integration tasks. Adapters and other add-on packages provide additional services that you use to interact with specific resources or applications. webMethods graphical implementation language, flow, enables you to quickly aggregate services into powerful sequences called flow services.

A Process Overview

Building a service is a process that involves the following basic stages:

Stage 1 Creating a new service on webMethods Integration Server.

During this stage, you create the new service on the webMethods Integration Server where you will do your development and debugging. For information about creating a flow service, see "Creating a New Flow Service" on page 203. For information about creating a Java service, see "Creating a Java Service" on page 310.

Stage 2 Adding logic to the new service.

During this stage, you specify the work that you want the service to perform.

If you are building a flow service, you add logic by inserting flow steps into the service. For more information, see "Building Flow Services" on page 199.

If you are building a Java service, you add logic to the source code for the service. For more information, see "Building Java Services" on page 303.

Stage 3 Declaring the service signature.

During this stage, you define the service's inputs and outputs. For information about this stage, see "About the Service Signature" on page 159.

Stage 4 Mapping pipeline data.

If you are building a flow service, during this stage, you route input and output variables between services that are invoked in the flow. For information about this stage, see "Mapping Data in Flow Services" on page 251.

Stage 5 Specifying the run-time parameters.

During this stage, you assign parameters that configure the run-time environment for this service. For information about this stage, see "About Service Run-Time Parameters" on page 165.

Stage 6 Configure the circuit breaker.

During this stage, you determine whether you need to enable and configure a circuit breaker for the service which can prevent a failure in one part of the system from

cascading to the rest of the system. For more information about this stage, see "Using a Circuit Breaker with a Service" on page 187.

Stage 7 Formatting service output.

During this stage you can create an output template to format the service output. For information about this stage, see "About Service Output Templates" on page 195 or refer to the *Dynamic Server Pages and Output Templates Developer's Guide*.

Stage 8 Debugging.

During this stage you can use the tools provided by Designer to run and debug your flow service. For information about this stage, see "Running Services" on page 403, "Debugging Flow Services" on page 429, and "Debugging Java Services" on page 455.

Note:
You can create templates with a set of pre-defined values for element properties. You can then apply the template when creating new instances of the element instead of setting the properties each time you create an element. For more information about the element property templates, see "Using Property Templates with Elements" on page 74.

Package and Folder Requirements

Before you create a new service, you must:

- **Make sure the package in which you want to create the service already exists.** If the package does not already exist, create it using Designer. For more information about creating a package, see "Creating a Package" on page 142.
- **Make sure the folder in which you want to create the service already exists and that you have Write ACL access to it.**If the folder does not already exist, create it using Designer. For information about creating folders, see "Creating New Elements" on page 41. For information about ACL permissions, see "Assigning and Managing Permissions for Elements" on page 79.

Once the package and folder are in place, use the **File > New** command to start the process of creating a new service. For details, see "Creating New Elements" on page 41.

About the Service Signature

Input and output parameters are the names and types of fields that the service requires as input and generates as output. Some systems refer to input and output parameters as "imports" and "exports." These parameters are also collectively referred to as a *signature*. You declare a signature for all types of services: flow services, Java services, and services written in other supported programming languages.

For example, a service that takes two string values—an account number (*AcctNum*) and a dollar amount (*OrderTotal*)—as input and produces an authorization code (*AuthCode*) as output, has the following input and output parameters:

Input Parameters		Output Parameters	
Name	**Data Type**	**Name**	**Data Type**
AcctNum	String	*AuthCode*	String
OrderTotal	String		

Although you are not required to declare input and output parameters for a service (the Integration Server will execute a service regardless of whether it has a specification or not), there are good reasons to do so:

- Declaring parameters makes the service's input and outputs visible to Designer. Without declared input and output parameters, you cannot:
  - Link data to and/or from the service using the Pipeline view.
  - Assign default input values to the service on the Pipeline view.
  - Validate the input and output values of the service at run time.
  - Log the input and output document fields of the service.
  - Run or debug the service in Designer and enter initial input values.
  - Generate skeleton code for invoking the service from a client.
- Declaring parameters makes the input and output requirements of your service known to other developers who may want to call your service from their programs.

For these reasons, it is strongly recommended that you make it a practice to declare a signature for every service that you create.

Designer supports several data types for use in services. Each data type supported by Designer corresponds to a Java data type and has an associated icon. When working in the editor, you can determine the data type for a field by looking at the icon next to the field name.

Guidelines for Specifying Input Parameters

When you define the input parameters for a service, keep the following points in mind:

- **Specify all inputs that a calling program must supply to this service.** For example, if a flow service invokes two other services, one that takes a field called *AcctNum* and another that takes *OrderNum*, you must define both *AcctNum* and *OrderNum* as input parameters for the flow service.

  Note:
  The purpose of declaring input parameters is to define the inputs that a calling program or client must provide when it invokes this flow service. You do not need to declare inputs that are obtained from within the flow itself. For example, if the input for one service in the

flow is derived from the output of another service in the flow, you do not need to declare that field as an input parameter.

- **When possible, use variable names that match the names used by the services in the flow.** Variables with the same name are automatically linked to one another in the pipeline. (Remember that variable names are case sensitive.) If you use the same variable names used by flow's constituent services, you reduce the amount of manual data mapping that needs to be done. When you specify names that do not match the ones used by the constituent services, you must use the Pipeline view to manually link them to one another.
- **Avoid using multiple inputs that have the same name.** Although Designer permits you to declare multiple input parameters with the same name, the fields may not be processed correctly within the service or by services that invoke this service.
- **Make sure the variables match the data types of the variables they represent in the flow.** For example, if a service in the flow expects a document list called *LineItems*, define that input variable as a document list. Or, if a service expects a Date object called *EmploymentDate*, define that input variable as an Object and apply the java.util.Date object constraint to it. For a complete description of the data types supported by Designer, see "Data Types" on page 1083.
- **Declared input variables appear automatically as inputs in the pipeline.** When you select the first service or MAP step in the flow, the declared inputs appear under **Pipeline In**.
- **Trigger services have specific input parameter requirements.** If you intend to use a service with a webMethods Messaging Trigger or a JMS trigger, make sure the input signature conforms to the requirements for each of those trigger types. For more information about creating webMethods Messaging Trigger, see "Creating a webMethods Messaging Trigger " on page 682. For more information about creating JMS triggers, see "Working with JMS Triggers" on page 629.

Important:
If you edit a cached service by changing the inputs (not the pipeline), you must reset the server cache. If you do not reset it, the old cached input parameters will be used at run time. To reset the service cache from Designer, select the service and then click the **Reset** button next to **Reset Cache** in the Properties view. To reset the service cache from Integration Server Administrator, select **Service Usage** under **Server** in the Navigation panel. Select the name of the service and an information screen for that service appears. Click **Reset Server Cache**.

Guidelines for Specifying Output Parameters

On the output side of the Input/Output tab you specify the variables that you want the service to return to the calling program or client. The guidelines for defining the output parameters are similar to those for defining input parameters:

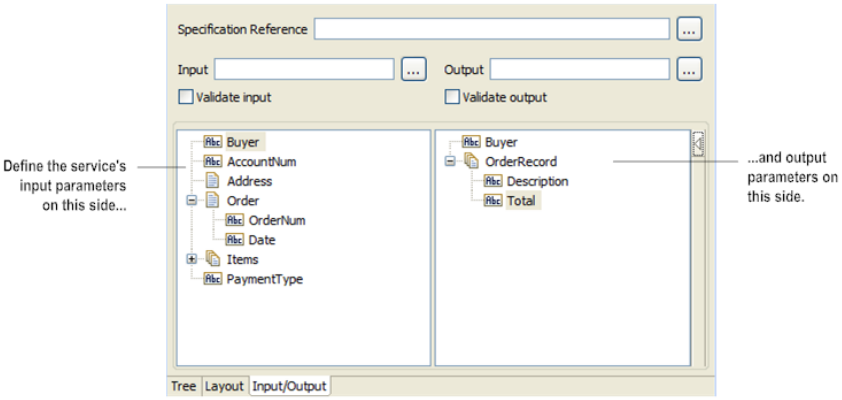
- **Specify all of the output variables that you want this service to return** to the calling program or client.
- **Make sure the names of output variables match the names used by the services** that produce them. Like input variables, if you do not specify names that match the ones produced by the flow's constituent services, you must use the Pipeline view to manually link them to one another.

- **Avoid using multiple outputs that have the same name.** Although Designer permits you to declare multiple output parameters with the same name, the fields may not be processed correctly within the service or by services that invoke this service.
- **Make sure the variables match the data types of the variables they represent in the service.** For example, if a service produces a String called *AuthorizationCode*, make sure you define that variable as a String. Or, if a service produces a Long object called *EmployeeID*, define that output variable as an Object and apply the java.lang.Long object constraint to it. For a complete description of the data types supported by a service, see "Data Types" on page 1083
- **Declared output variables appear automatically as outputs in the pipeline.**When you select the last service or MAP step in a flow, the declared output variables appear under **Pipeline Out.**

Declaring Input and Output Parameters

You declare the input and output parameters for a service using the Input/Output tab. On the left side of this tab, you define the variables that the service requires as input. On the right side, you define the variables the service returns to the client or calling program.

Input/Output tab



For a flow service, the input side describes the initial contents of the pipeline. In other words, it specifies the variables that this flow service expects to find in the pipeline at run time. The output side identifies the variables produced by the flow service and returned to the pipeline.

You can declare a service signature in one of the following ways:

- **Reference a specification.**A specification defines a set of service inputs and outputs. You can use a specification to define input and output parameters for multiple services. When you assign a specification to a service, you cannot add, delete, or modify the declared variables using the service's Input/Output tab.
- **Reference an IS document type.** You can use an IS document type to define the input or output parameters for a service. When you assign an IS document type to the Input or Output side of the Input/Output tab, you cannot add, modify, or delete the variables on that half of the tab.

- **Manually insert input and output variables.** Drag variables from the Palette view to the Input or Output sides of the Input/Output tab.

Using a Specification as a Service Signature

You can use a specification as the service signature. A *specification* is a "free-standing" IS element that defines a set of service inputs and outputs.

Keep the following points in mind when using a specification to define the service signature:

- A specification wholly defines the input and output parameters for a service that references it. This means that you cannot directly alter the service's input and output parameters through the service Input/Output tab. (Designer displays the parameters, but does not allow you to change them.) To make changes to the input and output parameters of the service, you must modify the specification (which affects all services that reference it) or detach the specification so you can manually define the parameters on the service's Input/Output tab.
- Any change that you make to the specification is automatically propagated to *all* services that reference that specification.
- If the specification resides in a different package than the service, you must set up a package dependency. For more information about package dependencies, see "About Package Dependencies" on page 149.

To assign a specification to a service

1. In the Package Navigator view, open the service to which you want to assign a specification.
2. Click the Input/Output tab.
3. In the **Specification Reference** field, type the fully qualified name of the specification, or click ... to select it from a list.
4. Click **OK**.

Using an IS Document Type to Specify Service Input or Output Parameters

You can use an IS document type as the set of input or output parameters for a service or specification. If you have multiple services with identical input parameters but different output parameters, you can use an IS document type to define the input parameters rather than manually specifying individual input fields for each service.

If the service uses an IS document type from a different package, you must set up a package dependency. For more information about package dependencies, see "About Package Dependencies" on page 149.

When an IS document type is assigned to the input or output of a service, you cannot add, delete, or modify the fields on that half of the Input/Output tab.

To use an IS document type as service input or output parameters

1. In the Package Navigator, double-click the service to which you want to assign the IS document type.
2. Click the Input/Output tab.
3. In the **Input** or **Output** field, type the fully qualified name of the IS document type or click to select it from a list. You can also drag an IS document type from the Package Navigator to the box below the **Validate input** or **Validate output** check boxes on the Input/Output tab.
4. Click **File > Save**.

Inserting Input and Output Parameters

You can define a service signature by dragging variables from the Palette view to the Input or Output side of the Input/Output tab.

To declare input and output parameters for a service

1. In the Package Navigator view, open the service for which you want to declare input and output parameters.
2. Click the Input/Output tab.
3. If the Palette view is not visible, display it by clicking on the right side of the editor.
4. In the Palette view, select the type of variable you want to define and drag it to the Input or Output side of the Input/Output tab.
5. Type a name for the variable and press ENTER.
6. With the variable selected, set variable properties and apply constraints using the Properties view.
7. If the variable is a document or document list, add child variables to define the content of the document or document list. Use to indent each member beneath the document or document list variable.

   Note:
   You can add a document reference to a service signature by selecting an IS document type in the Package Navigator view and dragging it to the Input/Output tab.
8. Click **File > Save**.

About Service Run-Time Parameters

As a developer of a service, you can use the Properties view to specify the following service behavior:

- **State of a service.** You can maintain whether or not you want the server to treat it as a "stateless" service at run time.
- **Caching of service results.** You can cache service results to improve the response time of stateless services.
- **Execution locale of a service.** You can set the type of locale in which the Integration Server executes at run time.
- **Creating a URL alias for a service.** You can create an alias for the path portion of the URL used to invoke a service.
- **Saving and restoring of the pipeline.** You can save the pipeline or restore a previously saved pipeline at run time.
- **XML format for the service input.** If the service receives an XML document, you can specify the format that Integration Server uses for the document when it passes the document to the service. The format you specify determines whether or notIntegration Server parses XML document before passing it to the service.

Important:
The run-time parameters should only be set by someone who is thoroughly familiar with the structure and operation of the selected service. Improper use of these options can lead to a service failure at run time and/or the return of invalid data to the client program.

Maintaining the State of Service

When a remote client opens a session on a webMethods Integration Server, the server automatically builds a session object for that client. The server uses this object to maintain specific information about the client requesting the service, such as user name and password. The server maintains the session object for the duration of the session (that is, until the client program explicitly closes the session on the server or the session times out due to client inactivity).

When you develop services in a language such as Java or C/C++, you can use the "put" method to write information to the session object. You might do this to store information that a sequence of services needs to maintain a connection to an external system.

A service that is an atomic unit of work (that is, one that is wholly self contained and not part of a multi-service transaction to an external system) does not to need to have its session object maintained when it is finished executing. For best performance, use stateful services if your Integration Server receives requests from repeating clients. The client can connect to Integration Server, be authenticated just once, and then issue many service invocations during the same session. Use stateless services if clients typically send a single invocation request to Integration Server at a time. Using a stateless service prevents the creation of sessions that will sit unused, taking up resources in Integration Server.

Specifying the Run-Time State for a Service

Configure run-time settings for a service in the Properties view.

To configure a service's run-time state

1. In the Package Navigator, open the service that you want to configure.
2. In the **Run time** category of the Properties view, do one of the following to set the **Stateless** property:
   - If the service is a self-contained, atomic unit of work and does not need access to state information, select **True**. The server will remove the client session immediately after the service executes, and no session information will be maintained for the service.
   - If the service is part of a multi-service transaction or if you are unsure of its state requirements, select **False**. The server will build and maintain a session object for this service.

   Important:
   Do not use the stateless option unless you are certain that the service operates as an atomic unit of work. If you are unsure, set the **Stateless** property in the **Run time** category to **False**.
3. Click **File > Save**.

About Service Caching

Caching is an optimization feature that can improve the performance of stateless services. When you enable caching for a service, Integration Server saves the entire contents of the pipeline after invoking the service in a local cache for the period of time that you specify. The pipeline includes the output fields explicitly defined in the cached service, as well as any output fields produced by earlier services in the flow. When the server receives subsequent requests for a service with the *same set of input values*, it returns the cached result to the client rather than invoking the service again.

Caching can significantly improve response time of services. For example, services that retrieve information from busy data sources such as high-traffic commercial web servers could benefit from caching. The server can cache the results for flows, Java services, and C/C++ services.

When Are Cached Results Returned?

When you enable caching for a service in Designer, Integration Server handles the cached results differently, depending on whether the service has input parameters. It is recommended that a cached service has input parameters.

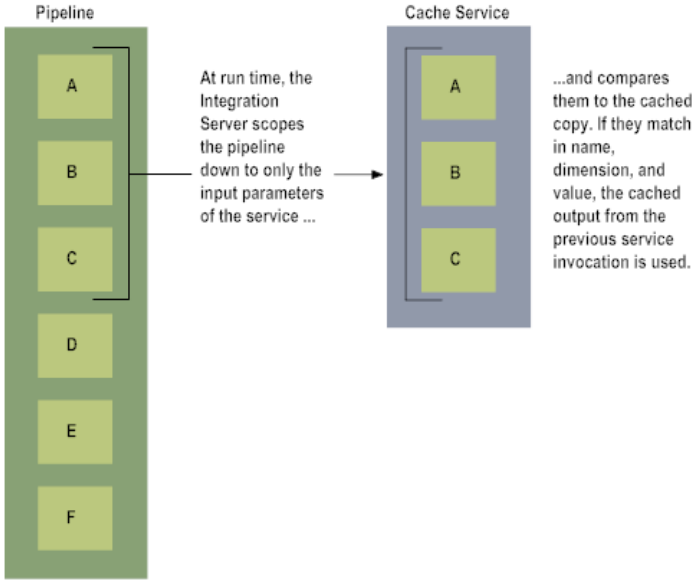
When a cached service has input parameters, at run time Integration Server scopes the pipeline down to only the declared input parameters of the service. Integration Server compares the

scoped-down inputs to the previously stored copy of the inputs. If a cached entry exists with input parameters that have the same values, Integration Server returns the cached results from the previous service invocation.

Note:

If a cached entry with input parameter values that are identical to the current invocation does not exist in the cache, Integration Server executes the service and stores the results in the cache.

Pipeline Inputs Are Compared to the Cached Copy at Run Time



When a cached service does not have input parameters (for example, a date/time service) and previous results do not exist in the cache, at run time Integration Server executes the service and stores the results. When the service executes again, Integration Server uses the cached copy. In other words, Integration Server does not use the run-time pipeline for the current service invocation; you will always receive cached results until the cache expires.

Important:
If a cached service input signature includes a Document Reference or Document Reference List variable and the referenced document type changes or is modified, you must reset the service cache. If you do not reset it, Integration Server uses the old cached input parameters at run time until such time as the cached results expire. You can reset the cache from Designer or Integration Server Administrator. For more information about resetting service cache from Integration Server Administrator, see *webMethods Integration Server Administrator's Guide*.

Types of Services to Cache

While caching service results can improve performance, not all services should be cached. You should never cache services if the cached results might be incorrect for subsequent invocations or if the service performs tasks that must be executed each time the service is invoked. Following are guidelines for you to consider when determining whether to cache the results for a service.

Services suited for caching:

- **Services that require no state information.** If a service does not depend on state information from an earlier transaction in the client's session, you can cache its results.
- **Services that retrieve data from data sources that are updated infrequently.** Services whose sources are updated on a daily, weekly, or monthly basis are good candidates for caching.
- **Services that are invoked frequently with the same set of inputs.** If a service is frequently invoked by clients using the same input values, it is beneficial to cache the results.

Services that you should *not* cache:

- **Services that perform required processing.** Some services contain processing that must be processed each time a client invokes it. For example, if a service contains accounting logic to perform charge back and you cache the service results, the server does not execute the service, so the service does not perform charge back for the subsequent invocations of the service.
- **Services that require state information.** Do not cache services that require state information from an earlier transaction, particularly information that identifies the client that invoked it. For example, you do not want to cache a service that produced a price list for office equipment if the prices in the list vary depending on the client who initially connects to the data source.
- **Services that retrieve information from frequently updated sources.** If a service retrieves data from a data source that is updated frequently, the cached results can become outdated. Do not cache services that retrieve information from sources that are updated in real time or near real time, such as stock quote systems or transactional databases.
- **Services that are invoked with unique inputs.** If a service handles a large number of unique inputs and very few repeated requests, you will gain little by caching its results. You might even degrade server performance by quickly consuming large amounts of memory.

Controlling a Service's Use of Cache

You use the properties in the Properties view to enable caching and to configure the way in which you want it to operate with the selected service. You use these settings to strike the right balance between data currency and memory usage. To gauge the effectiveness of your cache settings, you can monitor its performance by viewing service statistics with the Integration Server Administrator and then adjusting your caching values accordingly.

Note:
If you do not have administrator privileges on your Integration Server, work with your server administrator to monitor and evaluate your service's use of cache.

When returning results for a cached service, Integration Server returns a reference to the cached results instead of the actual value of the cached results. If a subsequent step in the service modifies the returned result, Integration Server changes the cached value as well, which affects all references to the cached value. If any other service uses the cached results, those services will begin using the updated cache value. To address this issue, you can do one of the following:

- Do not change the results of a cached service in a subsequent step in the flow service.

- Configure the ServiceResults cache to return the actual value instead of a reference, you need to modify the service results cache. For more information about changing the ServiceResults cache, see the *webMethods Integration Server Administrator's Guide*.

Important: Integration Server resets the cache for a service automatically whenever any edits are made to the service. However, if the input signature includes a document reference variable and the referenced document type changes, you must reset the service cache. If you do not reset it, Integration Server uses the old cached input parameters at run time until such time as the cached results expire. To reset the service cache from Designer, select the service and then click **Reset** next to Reset Cache in the Properties view. To reset the service cache from Integration Server Administrator, select **Service Usage** under **Server** in the Navigation panel. Select the name of the service and an information screen for that service appears. Click **Reset Server Cache**.

Specifying the Duration of Cached Results

Integration Server maintains results in cache for the period of time you specify in the **Cache expire** property on the Properties view. The expiration timer begins when the server initially caches a result, and it expires when the time you specify elapses. (The server does not reset the expiration timer each time it satisfies a service request with a cached result.) The minimum cache expiration time is one minute.

Note:
The cache may not be refreshed at the exact time specified in **Cache expire**. It may vary from 0 to 15 seconds, according to the cache sweeper thread. For details, see the watt.server.cache.flushMins setting in Integration Server.

Refreshing Service Cache by Using the Prefetch Option

You use the **Prefetch** property to specify whether or not you want the server to automatically refresh the cache for this service when it expires. If you set **Prefetch** to **True**, the server automatically re-executes the service (using the same set of inputs as before) to update its results in cache. This action also resets the cache expiration timer.

Keep the following points in mind when using **Prefetch**:

- Use **Prefetch** carefully. Overuse can quickly exhaust the memory available for cache.
- Do not use **Prefetch** with Java or C/C++ services that invoke access-controlled services. Such services will fail during prefetch because the embedded service will be invoked without the proper access privileges. To avoid this problem, enable **Prefetch** on the invoked services rather than on the Java or C/C++ services that call them.
- When you enable **Prefetch**, you must also set the **Prefetch activation** property to specify when the server should initiate a prefetch. This setting specifies the minimum number of times a cached result must be accessed (hit) in order for the server to prefetch results. If the server retrieves the cached results fewer times than specified in the **Prefetch activation** property, the server will not prefetch the service results when the cache expires.

- The cache may not be refreshed at the exact time the last hit fulfills the **Prefetch activation** requirement. It may vary from 0 to 15 seconds, according to the cache sweeper thread. For details, see the watt.server.cache.flushMins setting in Integration Server.

Configuring Caching of Service Results

Configure cache settings for a service in the Properties view.

To enable caching of pipeline contents after a service is invoked

1. In the Package Navigator, open the service for which you want to configure caching.
2. In the **Run time** category of the Properties view, set **Cache results** to **True**.
3. In the **Cache expire** field, type an integer representing the length of time (in minutes) that you want the results for this service to be available in cache.
4. If you want to use prefetch, set **Prefetch** to **True,** and then in the **Prefetch activation** property, specify the minimum number of hits needed to activate the use of prefetch.
5. Click **File > Save**.

Specifying the Execution Locale

When you create a service, you can set the locale property to indicate the locale policy in which the service executes at run time. The *locale policy* of a service refers to the language, regional, or cultural settings of a specific target market (the end user). Each locale consists of five sections: language, extended language, script, region, and variant.

Locales can influence the following:

- String display of numeric values and date/time values
- Parsing of dates and numbers from strings
- Default currency (pounds, Euros, dollars)
- Default measuring system (metric or customary)
- Default system resources (such as fonts, character encoding, etc.)
- Collation (sorting) of lists
- User interface/content language

Integration Server recognizes the following locale policies at run time:

- **Server locale** uses its default JVM locale.
- **User locale** uses the client locale.

- **Root locale** uses neutral or POSIX locale.
- **Null locale** uses no locale policy.

You can also configure Integration Server to recognize custom locales. By default, the service uses the null locale. That is, it uses no locale policy.

To specify the execution locale of a service

1. In the Package Navigator, open the service that you want to configure.
2. In the **Run time** category of the Properties view, do one of the following to specify the **Execution Locale** property:

Select...	To...
[$default] Default Runtime Locale	Use the server's default JVM locale.
[$user] Default User Locale	Use the client locale.
[$null] No Locale Policy	Use no locale policy.
[root locale]	Use the neutral or POSIX locale.
[<ISO code>] <Language>	Use a specific locale.
Open locale editor...	Define a custom locale.

3. If you selected **Open locale editor**, complete the following in the **Define Custom Locale** dialog box.

In this field...	Do the following...
Language	Select one of the ISO 639 codes that represent the language. (2- or 3-letter codes)
Extended Language	For future release.
Script	Optional. Select one of the 4-letter script codes in the ISO 15924 registry.
Region	Optional. Select one of the ISO 3166-2 country codes.
IANA Variant	Optional. Add or remove a variant code registered by the IANA.

4. Click **OK**. Integration Server will execute the service in the specified locale.

About URL Aliases for Services

Using a URL alias for a service is convenient because it saves you from specifying full path information for the service every time you have to enter the service URL. Also, if a service URL

has an alias, you can update the path information for the service without having to modify the alias. Another benefit to using aliases is the added security; they prevent the external world from seeing the service names in a URL.

You can create URL aliases for services from Designer and from Integration Server. Create an alias from Integration Server if you want to assign aliases to resources other than services, or if you want to assign more than one alias to a resource. See *webMethods Integration Server Administrator's Guide* for more information.

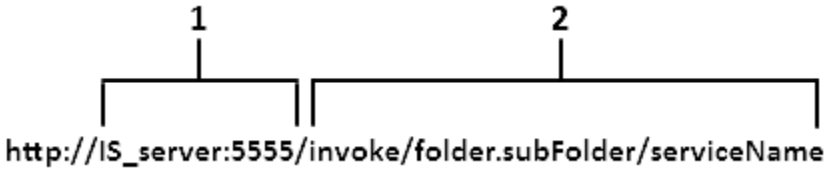
When you create a URL alias, you specify an alias for the path portion of the URL used to invoke a service. The path portion of the URL consists of the directive and the fully qualified service name.

To create an alias for a service URL, use the **HTTP URL alias** property in the Properties view. When you specify an alias in the **HTTP URL alias** property and save the service, Integration Server creates an HTTP path alias for the service URL. The target of the alias is the path that invokes the service. The path alias is the string that you entered in the property field.

The following sections provide examples that explain the substitution of the path portion of a service URL with a URL alias in a flow service and in a REST service.

URL Alias for a Flow Service

Consider the following example of the URL for a flow service:



Item	Description
1	Identifies the webMethods Integration Server on which the flow service you want to invoke resides.
2	Specifies the path portion of the URL for which the URL alias is to be generated. This portion includes the invoke directive "`invoke`." The path also identifies the folder in which the flow service resides and the name of the service to invoke. Separate subfolders with periods. These fields are case sensitive. Be sure to use the same combination of upper and lower case letters as specified in the folder name on webMethods Integration Server.

To create the URL alias for a flow service, replace the portion of the URL containing the invoke directive with an alias name in the **HTTP URL alias** property and save the service. For example, if the name of a flow service is folder.subFolder:serviceName, then the path to invoke the service is invoke/folder.subFolder/serviceName. If you enter "test" in the **HTTP URL alias** property and save the service, then the two following URLs will point to the same service:

http://IS_server:5555/invoke/folder.subFolder/serviceName

http://IS_server:5555/test

URL Alias for a REST Service (that uses the rest directive)

Important:
You can create a URL alias for a REST service that uses the rest directive, *not* a service that uses the restv2 directive.

Consider the following example of the URL for a REST service:

Item	Description
1	Identifies the webMethods Integration Server on which the REST service you want to invoke resides.
2	Specifies the path portion of the URL for the URL alias is to be generated. This portion includes the rest directive "`rest`." The path also identifies the REST resource folder in which the service resides. If the REST service resides in a subfolder within the folder specified in the example, then the path portion of the service URL is /rest/folder/subFolder/resource.

If you consider an example REST service named folder.subFolder:_put, the path to invoke the service is rest/folder/resource/. The path need not include the REST service name because Integration Server identifies the service to invoke based on the HTTP method specified in the client request.

Similar to the flow service URL alias, if you specify "test" as the **HTTP URL alias** property and save the REST service, then the two following URLs will point to the same service:

http://IS_server:5555/rest/folder/Resource/

http://IS_server:5555/test

Creating a Path Alias for a Service

Use the **HTTP URL alias** property in the Properties view to create a path alias to use when invoking the service in a URL.

When creating a path alias for a service, keep the following in mind:

- When you add, edit, or delete an **HTTP URL alias** property in a service, the property is automatically updated on the Integration Server when the service is saved.

- Integration Server stores the **HTTP URL alias** information in the node.ndf file of the service. Because the property is encoded in the node.ndf file, it is propagated across servers through package replication.
- URL aliases for services are saved in memory on the Integration Server. The server checks for URL aliases before processing the client HTTP requests.
- When specifying the alias URL, you must spell it exactly as it is defined on the server. Alias URLs are case sensitive.
- URL alias strings must be unique on the server. The server cannot register a URL alias if one already exists with the same name on the server, even if it belongs to a different package.
- To troubleshoot alias URLs that cannot register, open the **Package > Management > *PackageName*** screen in the Integration Server Administrator and view the information under **Load Warnings**. The alias URLs that generated warnings when the package was installed are displayed here with a brief explanation.

To create an alias for a service URL

1. In the Package Navigator, open the service that you want to configure.
2. In the **Run time** category of the Properties view, next to the **HTTP URL alias** property, enter an alias string for the URL that will invoke the service.

   Important:
   Do not use reserved characters in the URL alias string. Alias strings that contain reserved characters are invalid and will not work.
3. On the **File** menu, click **Save**.

Automatically Saving or Restoring the Pipeline at Run Time

Use the **Pipeline debug** property to save or restore the pipeline for a flow service at run time. The ability to save and restore the pipeline is especially useful when you are testing and debugging the service.

If you set **Pipeline debug** to **Save**, Integration Server saves the entire contents of the pipeline to a file just before the service executes. You can use the data in the saved pipeline file to debug and troubleshoot the service or to restore the pipeline. For more information about debugging and troubleshooting services, see "Saving and Restoring the Flow Service Pipeline while Debugging" on page 447.

You can use the **Pipeline debug** property to specify that Integration Server restore the pipeline for the service automatically at run time. When you restore the service pipeline using this property, Integration Server either merges or overwrites the pipeline using the contents from a previously saved pipeline file.

Restoring the service pipeline is useful when debugging or troubleshooting the service or you just want to inspect the values of the pipeline file. For more information about debugging and troubleshooting services, see "Saving and Restoring the Flow Service Pipeline while Debugging" on page 447.

Important:
The pipeline debug options you select can be overwritten at run time by the value of the watt.server.pipeline.processor property set in the server configuration file. This property globally enables or disables the Pipeline debug settings. The default enables the Pipeline debug feature on a service-by-service basis. For more information on setting properties in the server configuration file, see *webMethods Integration Server Administrator's Guide*.

Configuring Saving or Restoring of the Service Pipeline at Run Time

To configure saving or restoring of the service pipeline at run time

1. In the Package Navigator, open the service that you want to configure.
2. Under the **Run time** category in the Properties view, select one of the following from the **Pipeline Debug** property:

Select...	To...
None	Run the service without saving or restoring the pipeline. This is the default.
Save	Save the pipeline to a file when the service executes. When the service executes, the contents of the pipeline are saved to a file on webMethods Integration Server. The file is saved as *folderName.serviceName*.xml in the *IntegrationServer_directory*\instances*instance_name*\pipeline directory. If the file does not exist, the service creates it. If the file already exists, the service overwrites it.
Restore (Override)	Restore the pipeline from a file when the service executes. When the service executes, the server loads the pipeline file, *folderName.serviceName*.xml, from the *IntegrationServer_directory*\instances*instance_name*\pipeline directory. The server will throw an exception if the pipeline file does not exist or cannot be found.
Restore (Merge)	Merge the pipeline with one from a file when the service executes. When this option is selected and the input parameters in the file match the input parameters in the pipeline, the values defined in the file are used in the pipeline. If there are input parameters in the pipeline that

Select...	To...
	are not matched in the file, the input parameters in the pipeline remain in the pipeline. When the service executes, the server loads the pipeline file, *folderName.serviceName*.xml, from the *IntegrationServer_directory*\instances*instance_name*\pipeline directory. The server will throw an exception if the pipeline file does not exist or cannot be found.

Specifying the Default Format for an XML Document Received by the Service

For a service that receives an XML document, you can specify the format for the XML document that Integration Server passes to the service. The XML format that you select determines:

- Whether Integration Server parses the XML document automatically before passing it to the service
- Which XML parser Integration Server uses. When parsing an XML document, Integration Server uses either the legacy XML parser or the enhanced XML parser. For more information about the XML parsers, see *webMethods Integration Server Administrator's Guide*.
- The name and data type of the variable that Integration Server adds to the pipeline with the contents of the XML document

The **Default xmlFormat** property specifies the default handling for XML documents received by the service. Keep the following points in mind when setting the **Default xmlFormat** property value for a service:

- You can specify a default XML format for flow services and Java services only. The **Default xmlFormat** property is not available for C/C++ services, .NET services, or web service connectors.
- The default XML format specified for a service by the **Default xmlFormat** property can be overridden by the value of the xmlFormat argument in the URL of an individual client request. However, the client request should specify the xmlFormat argument only when it is recommended in the documentation for the service. A client should specify the xmlFormat only when knowing how the service will respond. For more information see "Submitting and Receiving XML via HTTP" on page 888
- The XML format determines whether or not Integration Server parses the document. If parsing is not needed, it can unnecessarily slow down the execution of a service. For example, an application might handle the XML as a simple String. In this case, the automatic parsing is unnecessary and should be avoided.
- Make sure the input signature of the service contains an input parameter that matches the variable name and data type that Integration Server produces for the default format.

To specify the default XML format for a service

1. In the Package Navigator view of the Service Development perspective, open the service for which you want to set the default XML format.

2. In the **Run time** category in the Properties view, next to **Default xmlFormat**, specify one of the following:

Select...	To specify that the default XML format is...
<blank>	Determined by the value of the watt.server.http.xmlFormat server configuration parameter. This is the default. For more information about the watt.server.http.xmlFormat server configuration parameter, see *webMethods Integration Server Administrator's Guide*.
bytes	A byte array. Integration Server passes the XML document directly to the service as a byte array without parsing the XML. Integration Server places the byte array in the input pipeline of the target service in a variable named *xmlBytes*.
enhanced	A node parsed by the enhanced XML parser. Integration Server parses the XML automatically using the enhanced XML parser. Integration Server uses the default options specified for enhanced XML parsing on the **Settings > Enhanced XML Parsing** page in Integration Server Administrator. Integration Server passes the XML document to the target service as an enhanced node that implements the org.w3c.dom.Node interface. Integration Server places the node in the input pipeline of the target service in a variable named *node*.
node	A node parsed by the legacy XML parser. Integration Server parses the XML automatically using the legacy parser and passes it to the target service as a node of type com.wm.lang.xml.Node. Integration Server places the node in the input pipeline of the target service in a variable named *node*
stream	An InputStream. Integration Server passes the XML document directly to the service as an XML stream without parsing the XML. Integration Server places the XML stream in the input pipeline of the target service as an InputStream named *xmlStream*.

3. Click **File > Save**.

Configuring HTTP Methods Allowed for a Service

For security purposes, you can configure Integration Server services to process client requests containing only certain HTTP methods. The HTTP methods that you configure as allowed for a service override the methods that are allowed for any REST resource corresponding to the service. Integration Server issues a "405 Method Not Allowed" error in response to any request containing a method that is not configured as allowed for a service.

To configure the HTTP methods allowed for a service

1. In the **Run time** category in the Properties view, click ... next to **Allowed HTTP Methods**.

   The HTTP Methods dialog box appears.

2. Select the HTTP methods you want to configure for the service, and click **OK** to save your changes.

   Important:
   If the service already has REST resources configured, Designer displays a warning message if you change the selection of the allowed HTTP methods to exclude any method used in the configuration of the resources. In such a situation, any client request invoking the excluded method will fail.

   Therefore, you must ensure that the set of HTTP methods configured for a REST resource is always a subset of the methods allowed for the underlying service.

About Automatic Service Retry

You can set Integration Server so that it retries a service if the service fails because of a transient error. A *transient error* is an error that arises from a temporary condition that might be resolved or restored quickly, such as the unavailability of a resource due to network issues or failure to connect to a database. The service might execute successfully if Integration Server waits a short interval of time and then retries the service.

To set up automatic service retry:

- Configure each service you want Integration Server to automatically retry by specifying:
  - Retry interval, which defines how long Integration Server waits before trying to re-execute the service.
  - Maximum retry attempts, which is the number of times you want Integration Server to try to re-execute the service.

  For more information, see "Configuring Service Retry" on page 179.
- Configure the watt.server.invoke.maxRetryPeriod server configuration parameter to set the maximum retry period. The maximum retry period is the total amount of time that can elapse

for Integration Server to make all possible retry attempts. For more information, see "About the Maximum Retry Period" on page 179.

- Build your service so that it throws an ISRuntimeException when a transient error occurs. For more information, see "Building Services that Retry" on page 903.

At run time, when a service that is coded for retry determines that a transient error occurred, it throws an ISRuntimeException to signal Integration Server to retry the service. When a service throws an ISRuntimeException, Integration Server waits the length of the retry interval and then re-executes the service using the original input pipeline passed to the service. Integration Server continues to retry the service until the service executes successfully or Integration Server makes the maximum number of retry attempts. If the service throws an ISRuntimeException during the final retry attempt, Integration Server treats the last failure as a service error. The service ends with a service exception.

Integration Server generates the following journal log message between retry attempts:

```
[ISS.0014.0031V3] Service serviceName failed with ISRuntimeException.
Retry x of y will begin in retryInterval milliseconds.
```

Note:
If service auditing is also configured for the service, Integration Server adds an entry to the service log for each failed retry attempt. Each of these entries will have a status of "Retried" and an error message of "Null". However, if Integration Server makes the maximum retry attempts and the service still fails, the final service log entry for the service will have a status of "Failed" and will display the actual error message.

About the Maximum Retry Period

Integration Server uses the same server thread for the initial service execution and the subsequent retry attempts. Integration Server returns the thread to the server thread pool only when the service executes successfully or the retry attempts are exhausted. To prevent the execution and re-execution of a single service from monopolizing a server thread for a long time, Integration Server enforces a *maximum retry period* when you configure service retry properties. The maximum retry period indicates the total amount of time that can elapse if Integration Server makes the maximum retry attempts. By default, the maximum retry period is 15,000 milliseconds (15 seconds).

When you configure service retry, Integration Server verifies that the retry period for that service will not exceed the maximum retry period. Integration Server determines the retry period for the service by multiplying the maximum retry attempts by the retry interval. If this value exceeds the maximum retry period, Designer displays an error indicating that either the maximum attempts or the retry interval needs to be modified.

Note:
The watt.server.invoke.maxRetryPeriod server parameter specifies the maximum retry period. To change the maximum retry period, change the value of this parameter.

Configuring Service Retry

When configuring service retry, keep the following points in mind:

- You can configure retry attempts for flow services, Java services, and C services only.
- Only top-level services can be retried. That is, a service can be retried only when it is invoked directly by a client request. The service cannot be retried when it is invoked by another service (that is, when it is a nested service).
- If a service is invoked by a trigger (that is, the service is functioning as a trigger service), Integration Server uses the trigger retry properties instead of the service retry properties.
- Unlike webMethods Messaging Triggers, you cannot configure a service to retry until successful.
- To catch a transient error and re-throw it as an ISRuntimeException, the service must do one of the following:
  - If the service is a flow service, the service must invoke pub.flow:throwExceptionForRetry. For more information about the pub.flow:throwExceptionForRetry, see the *webMethods Integration Server Built-In Services Reference*.
  - If the service is written in Java, the service can use com.wm.app.b2b.server.ISRuntimeException ( ). For more information about constructing ISRuntimeExceptions in Java services, see *webMethods Integration Server Java API Reference* for the com.wm.app.b2b.server.ISRuntimeException class.
- The service retry period must be less than the maximum retry period. For more information, see "About the Maximum Retry Period" on page 179.

To configure service retry

1. In Package Navigator view, open the service for which you want to configure service retry.
2. Under **Transient error handling** in the Properties view, in the **Max retry attempts** property, specify the number of times Integration Server should attempt to re-execute the service. The default is 0, which indicates that Integration Server does not attempt to re-execute the service.
3. In the **Max interval** property, specify the number of milliseconds Integration Server should wait between retry attempts. The default is 0 milliseconds, which indicates that Integration Server re-executes the service immediately.
4. Click **File > Save**.

Tip:
You can invoke the pub.flow:getRetryCount service to retrieve the current retry count and the maximum specified retry attempts. For more information about this service, see the *webMethods Integration Server Built-In Services Reference*. For more information about building a service that retries, see "About Automatic Service Retry" on page 178.

About Service Auditing

Service auditing is a feature in Integration Server that you can use to track which services executed, when services started and completed, and whether services succeeded or failed. You perform

service auditing by analyzing the data stored in the service log. The service log can contain entries for service start, service end, and service failure. The service log can also contain a copy of the input pipeline used to invoke the service as well as select fields from input and output service signatures. At run time, services generate audit data at predefined points. Integration Server captures the generated audit data and stores it in the service log. If the service log is a database, you can re-invoke services using the webMethods Monitor.

Note:
When Integration Server logs an entry for a service, the log entry contains the identify of the server that executed the service. The server ID in the log entry *always* uses the Integration Server primary port, even if a service is executed using another (non-primary) Integration Server port.

Each service has a set of auditing properties located in the **Audit** category on the service's Properties view. These properties determine when a service generates audit data and what data is stored in the service log. For each service, you can decide:

- Whether the service should generate audit data during execution. That is, do you want the service to generate audit data to be captured in the service log? If so, you must decide whether the service will generate audit data every time it executes or only when it is invoked directly by a client request (HTTP, FTP, SMTP, etc.) or a trigger.
- The points during service execution when the service should generate audit data to be saved in the service log. You might want a service to produce audit data when it starts, when it ends successfully, when it fails, or a combination of these.
- Whether to include a copy of the service input pipeline in the service log. If the service log contains a copy of the input pipeline, you can use the webMethods Monitor to perform more extensive failure analysis, examine the service's input data, or re-invoke the service.

Keep in mind that generating audit data can impact performance. Integration Server uses the network to send the audit data to the service log and uses memory to actually save the data in the service log. If a large amount of data is saved, performance can be impacted. When you configure audit data generation for services, you should balance the need for audit data against the potential performance impact.

Note:
The service log can be a flat file or a database. If you use a database, the database must support JDBC. You can use Integration Server to view the service log whether it is a flat file or a database. If the service log is a database, you can also use the webMethods Monitor to view audit data and re-invoke the service. Before you configure service auditing, check with your Integration Server Administrator to learn what kind of service log exists. For more information about the service log, see the *webMethods Audit Logging Guide*.

Service Auditing Use Cases

Before you set properties in the **Audit** category on the Properties view, decide what type of auditing you want to perform. That is, decide what you want to use the service log for. The following sections describe four types of auditing and identify the **Audit** properties you would select to be able to perform that type of auditing.

Error Auditing

In error auditing, you use the service log to track and re-invoke failed services. To use the service log for error auditing, services must generate audit data when errors occur, and the Integration Server must save a copy of the service's input pipeline in the service log.

With webMethods Monitor, you can only re-invoke top-level services (those services invoked directly by a client or by a webMethods Messaging Trigger). Therefore, if your intent with error auditing is to re-invoke failed services, the service needs to generate audit data only when it is the top-level service and it fails.

To make sure the service log contains the information needed to perform error auditing, select the following **Audit** properties.

For this property...	Select this option...
Enable auditing	**When top-level service only** **Note:** If you want to be able to audit all failed invocations of this service, select **Always**.
Include pipeline	**On errors only**
Log on	**Error only**

To use the service log for error auditing, use a database for the service log.

Service Auditing

When you perform service auditing, you use the service log to track which services execute successfully and which services fail. You can perform service auditing to analyze the service log and determine how often a service executes, how many times it succeeds, and how many times it fails. To use the service log for service auditing, services need to generate audit data after execution ends.

To make sure the service log contains the information needed to perform service auditing, select the following **Audit** properties.

For this property...	Select this option...
Enable auditing	**When top-level service only**
Log on	**Error and success**
Include pipeline	**Never** **Note:**

For this property...	Select this option...
	Configure a service to save a copy of the input pipeline only if you intend to re-invoke the service using the resubmission capabilities of the webMethods Monitor.

To use the service log for service auditing, you can use either a flat file or a database as the service log.

Auditing for Recovery

Auditing for recovery involves using the service log to track executed services and service input data so that you can re-invoke the services. You might want to audit for recovery in the event that a resource experiences a fatal failure, and you want to restore the resource to its pre-failure state by resubmitting service invocations.

When auditing for recovery, you want to be able to resubmit failed and successful services. The service log needs to contain an entry for each service invoked by a client request or a trigger. The service log also needs to contain a copy of each service's input pipeline.

To use the service log to audit for recovery, select the following **Audit** properties.

For this property...	Select this option...
Enable auditing	**When top-level service only**
Log on	**Error and success**
Include pipeline	**Always**

To use the service log to audit for recovery, use a database for the service log.

Auditing Long-Running Services

If a service takes a long time to process, you might want to use the service log to verify that service execution started. If the service log contains a start entry for the service but no end or error entry, then you know that service execution began but did not complete. To enable auditing of long-running services, select the **Error, success, and start** option for the **Log on** property.

Note:
Typically, you will audit long-running services in conjunction with error auditing, service auditing, or auditing for recovery.

Configuring Service Auditing

When you configure auditing for a service, you can determine if and when a service generates audit data and whether the service log includes a copy of the service's input pipeline or select

fields. Make sure that you select options that will provide the service log with the audit data you require.

Keep the following points in mind when configuring service auditing:

- Before you select options for generating audit data, check with your Integration Server Administrator to determine what kind of service log exists. A service log can be a flat file or a database.
- The options you select in the **Audit** category of the Properties view can be overwritten at run time by the level set for the Service Logger in Integration Server. View the Service Logger level on the **Setting > Logging > View Service Logger Details** page of Integration Server Administrator.
- The service generates audit data only when it satisfies the selected option under **Enable auditing** and the selected option in the **Log on** property. For example, if **When top-level service only** is selected and the service is not the root service in the flow service, it will not generate audit data.
- The pipeline data saved in the service log is the state of the pipeline just before the invocation of the service. It is not the state of the pipeline at the point the service generates audit data.
- Including the pipeline in the service log is useful only when the service log is a database. Integration Server cannot save the pipeline to a flat file service log.
- When a service generates audit data, it also produces an audit event. If you want the audit event to cause another action to be performed, such as sending an e-mail notification, write an event handler. Then subscribe the event handler to audit events. For more information about events and event handlers, see "Subscribing to Events" on page 865
- If you want audit events generated by a service to pass a copy of the input pipeline to any subscribed event handlers, set **Include pipeline** to **On errors only** or **Always**.
- Integration Server can also log select fields from the service signature. Logged fields can be viewed in the webMethods Monitor. For information about field logging, see "Logging Input and Output Fields" on page 185.
- You can associate a custom value with an auditing context. The custom value can be used to search for service audit records in the webMethods Monitor. For information about creating and logging custom values for auditing contexts, see "Assigning a Custom Value to an Auditing Context" on page 187.
- To configure service auditing, you must have write access to the service and own the lock on the service or have it checked out.
- For detailed information about the Audit properties, see "Audit Properties" on page 1029.

To configure service auditing

1. In the Package Navigator, double-click the service for which you want to configure service auditing.

2. In the **Audit** category of the Properties view, select an **Enable auditing** option to indicate when you want the service to generate audit data.

3. For **Log on**, select an option to determine when the service generates audit data.

4. For **Include pipeline**, select an option to indicate when Integration Server should include a copy of the input pipeline in the service log.

5. Click **File > Save**.

Logging Input and Output Fields

You can select input and output fields from service signatures for logging. Values of logged fields can be viewed in webMethods Monitor. You can also create aliases for the logged fields, which makes them easier to locate in webMethods Monitor.

When Integration Server logs data for a field depends on the value of the **Log on** property. To use the Service logger to audit for both input and output service signatures, select the following **Log On** properties:

Log on...	Data logged at the start of the service...	Data logged at the end of the service...
Error	None	Input and output data
Error, Success	None	Input and output data
Error, Success, Start	Input data	Output data

Selecting Input or Output Fields for Logging

When selecting fields for logging, keep the following points in mind:

- Input and output parameters must be defined for the service on the Input/Output tab.
- Audit logging must be enabled for the service. The **Enable auditing** property for the service must be set to **When top-level service only** or **Always.**
- The Integration Server Service logger must be enabled. The Service logger must be configured to write to a database so you can use the webMethods Monitor to view audit data. You may have to check with your Integration Server Administrator to learn how the Service logger is configured. For more information about the enabling and configuring the Service logger, see the *webMethods Audit Logging Guide*.
- You can create the same alias for more than one field, but this is not recommended. Having the same alias might make monitoring the fields at run time difficult.

To select service input or output fields for logging

1. In the Package Navigator view, open the service for which you want to select fields for logging.

2. Click the Logged Fields tab.

3. On the Logged Fields page, click **Expand** to expand the **Inputs** and **Outputs** trees to display the fields available in the service signature.

4. Select the check boxes next to the fields you want to log.

5. If you want to define an alias for a field, type an **Alias** name.

   The alias defaults to the name of the selected field, but it can be modified to any alias for viewing in webMethods Monitor.

Logged Field Data Types in JDBC

When the value for a logged field is written to the database, it is saved as one of the JDBC data types: VARCHAR, FLOAT or DATE. Integration Server converts the data type for each logged field as follows:

Data Type in Input or Output Document	JDBC Type in Database
String, String List or Table	VARCHAR
Object or Object List, Java wrapper type:	
java.lang.Boolean	VARCHAR
java.lang.Byte	VARCHAR
java.lang.Character	VARCHAR
java.lang.Double	FLOAT
java.lang.Float	FLOAT
java.lang.Integer	FLOAT
java.lang.Long	FLOAT
java.lang.Short	FLOAT
java.util.Date	DATE
byte[]	VARCHAR
UNKNOWN	VARCHAR

Integration Server calls the toString() method on objects that do not have a defined Java wrapper type. If you are logging one of your own types and you implement the toString() method, the server saves the value returned by your implementation to the audit log. If you do not supply a toString implementation, the server saves the output of java.lang.Object.toString() to the database.

Assigning a Custom Value to an Auditing Context

You can assign a custom value to an auditing context. If Integration Server is configured to write service audit data to a database, you can use the custom value as search criteria to locate and view all corresponding service audit records. You search logged audit data using the webMethods Monitor.

To write custom values for the current auditing context to the server log, use the Integration Server built-in service pub.flow:setCustomContextID. For instructions about using this service, see the *webMethods Integration Server Built-In Services Reference*. For information about using the webMethods Monitor, see the webMethods Monitor documentation.

Using a Circuit Breaker with a Service

Circuit breaker is an established design pattern that applications implement to prevent a failure in one part of the system from cascading to the rest of the system. In an architecture with distributed applications, such as microservices, many services call other services running on remote servers. If the remote service is unavailable or the network is slow, the calling service may wait until a timeout occurs. During this time, the calling service continues to consume critical resources such as server threads and memory. If multiple services call the unresponsive or failing remote service, the impact of the remote service cascades throughout all the calling services, causing even more resources to be consumed and affected by a single failing service. Implementing a circuit breaker on the call to the remote service can prevent the impact of the failing or unresponsive service or network latency from cascading throughout the system.

The circuit breaker design pattern works much like an electrical circuit breaker which is intended to “trip” or open the circuit when failure is detected. This prevents the flow of electrical current through the circuit. After a time delay, the electrical circuit breaker resets and closes the circuit, which causes the flow of electricity to resume.

In a software application, a circuit breaker functions as a proxy that executes the remote services and monitors the remote service for failures. A failure can be an exception and/or a timeout. When the number of failures meets a predetermined threshold within a specified time period, the circuit breaker “trips” or opens the circuit. Subsequent requests for the service end with an error or result in execution of an alternative service. After a reset period elapses, the circuit breaker sets the circuit state to half-open and executes the next request for the service. By allowing a single request to execute and causing other requests to wait, the circuit breaker gauges the health of the service. Upon success of the service, the circuit breaker closes circuit and waiting requests proceed. However, if the service ends with another failure, the circuit breaker re-opens the circuit.

Circuit breakers can be especially useful in systems with a microservices architecture as these systems often feature a large number of distributed components. By configuring a circuit breaker on the invocation of the remote service, you can limit the impact the abnormal behavior of a remote service on other micorservices and critical resources in your system.

Note:
The circuit breaker feature is available by default for a service that resides in a Microservices Container. To use the circuit breaker feature with Integration Server, your Integration Server must have additional licensing. Note: In addition to the licensing requirement, to use the circuit

breaker functionality in version 10.1, you must install the following fixes: ESB_10.1_Fix2 and IS_10.1_Core_Fix2.

Considerations for Configuring a Circuit Breaker for a Service

The purpose of a using a circuit breaker with a service is to protect your application from failures of that service by monitoring the service for consistent failures and then preventing execution of the service until success is likely. Circuit breakers can be useful for services that communicate with distributed applications such as a service that connects to an application in the cloud or a database. By configuring a circuit breaker for the service, the continued inability to connect to the cloud or a database results in failure events which eventually result in the opening of the circuit.

When you configure a circuit breaker for a service, you must make the following key decisions.

- What constitutes a failure event? An exception, a timeout, or both?
- How many failure events must occur within a set time period before the circuit opens?
- What action does the circuit breaker take upon receiving a request for the service with the open circuit?
- How long does the circuit breaker wait before attempting to execute the service? That is, what is the circuit reset period after which the circuit state changes from open to half-open?

For more information about circuit breaker behavior and concepts, see *Developing Microservices with webMethods Microservices Container*

Keep the following information in mind when configuring a circuit breaker for a service.

- The circuit breaker feature is available by default for a service that resides in a Microservices Container. To use the circuit breaker feature with Integration Server, your Integration Server must have additional licensing. Note: In addition to the licensing requirement, to use the circuit breaker functionality in version 10.1, you must install the following fixes: ESB_10.1_Fix2 and IS_10.1_Core_Fix2.
- You can configure a circuit breaker for a user-defined service only. That is, you cannot configure a circuit breaker for services in packages provided by Software AG such WmPublic or WmRoot.
- Using a circuit breaker with a service may impact service performance. However, the benefits of using a circuit breaker may outweigh the performance impact.
- If the circuit breaker for a service considers a timeout to be a failure event and you want circuit breaker to attempt to cancel the thread executing the service, you must configure the thread kill functionality on the Integration Server that hosts the service. Specifically, you must set the watt.server.threadKill.enabled and the watt.server.threadKill.interruptThread.enabled server configuration parameters to true. For more information about the thread kill functionality, including configuration information and limitations, see *webMethods Integration Server Administrator's Guide*.
- When configuring a service to use circuit breaker and transient error handling, also known as service retry, keep in mind that circuit breaker could open the circuit before completing all of the retry attempts. The circuit breaker handles any subsequent retry attempts as it would any

request for the service. For more information about how a circuit breaker works, see *Developing Microservices with webMethods Microservices Container*.

- When specifying an alternate service to invoke when a service has an open circuit, do not create a circular reference in which the service with the configured circuit breaker calls itself.

Configuring a Circuit Breaker for a Service

Before you configure a circuit breaker for a service make sure you understand how circuit breaker works in Integration Server and review the guidelines in "Considerations for Configuring a Circuit Breaker for a Service" on page 188

To configure a circuit breaker for a service

1. In Package Navigator, open the service for which you want to configure a circuit breaker.
2. In the **Circuit Breaker** category of the Properties view, next to **Enabled**, specify one of the following:

Specify...	To...
True	Enable a circuit breaker for this service.
False	Disable a circuit breaker for this service. This is the default.

3. In the **Failure event** field, specify the events that the circuit breaker considers to be a failure event,

Select...	To...
Exception only	Indicate that a failure event occurs only when the service ends with an exception. This is the default.
Timeout only	Indicate that a failure event occurs only when the service execution time exceeds the **Timeout period** property value.
Exception or Timeout	Indicate that a failure event occurs when the service ends with an exception or the service execution time exceeds the **Timeout period** property.

4. If the circuit breaker treats a timeout as a failure event, configure the following information:

For this property...	Specify...
Timeout period	The number of seconds that service execution can take before being considered a timeout failure event. If the timeout period elapses before service execution completes, the circuit breaker considers a timeout failure event to have occurred. The default is 60 seconds. You must specify a timeout period greater than 0.
Cancel thread on timeout	Whether the circuit breaker gracefully attempts to cancel the thread executing the service when the timeout period elapses causing the timeout failure event. Canceling a thread can free up resources held by the thread.

Select...	To...
True	Indicate that circuit breaker attempts to cancel the thread executing the service if the timeout period elapses before the service execution completes.
False	Indicate that circuit breaker does not attempt to cancel the service execution thread if the timeout period elapses before service execution completes. This is the default.

5. In the **Failure threshold** field, specify the number of failure events that cause the circuit to open if all the events occur within the failure period. The default is 5.

6. In the **Failure period** field, specify the length of time, measured in seconds, during which the number of failure events equal to the failure threshold causes the circuit to open. The default is 60 seconds.

7. In the **Circuit open action** field, specify how circuit breaker responds to a request for this service when the circuit is open.

Select...	To...
Throw exception	Indicate that circuit breaker responds to a service request by throwing the exception that caused the circuit to open. This is the default.
Invoke service	Indicate that circuit breaker responds to a service request by executing the alternate service specified in the **Circuit open service** property.

8. If you set the **Circuit open action** field to **Invoke service**, in the **Circuit open service** field, specify the fully qualified name of the alternate service that circuit breaker invokes upon receiving a request when the circuit is open. For more information about building a service for use with an open circuit, see *Developing Microservices with webMethods Microservices Container*.

9. In the **Circuit reset period** field, specify the length of time, measured in seconds, for which the circuit remains in an open state. The default is 300 seconds.

   During the reset period, the circuit breaker responds to requests to invoke the service as specified by the **Circuit open action** property. When the reset period elapses, the circuit breaker places the circuit in a half-open state. The next request for the service results in service execution, after which the circuit breaker either closes or re-opens the circuit.

10. Click **File > Save**.

About Universal Names for Services or Document Types

Every service and document type on a webMethods Integration Server has a universal name in addition to its regular webMethods name. A *universal name* is a unique public identifier that external protocols (such as SOAP) use to reference a service or document type on an Integration Server.

The structure of a universal name is the same as the structure of a QName in an XML namespace and consists of two parts: a *namespace name* and a *local name*.

- The namespace name is a qualifier that distinguishes a webMethods service from other resources on the Internet. For example, there might be many resources with the name AcctInfo. A namespace name distinguishes one AcctInfo resource from another by specifying the name of the collection to which it belongs, similar to the way in which a state or province name serves to distinguish cities with the same name (for example, Springfield, *Illinois*, versus Springfield, *Ontario*).

  Like namespaces in XML, the namespace portion of a universal name is expressed as a URI. This notation assures uniqueness, because URIs are based on globally unique domain names.

  The namespace portion of the universal name can consist of any combination of characters that form a valid absolute URI (relative URIs are not supported). For example, the following are all valid namespace names:

  http://www.gsx.com

  http://www.gsx.com/gl/journals

  http://www.ugmed.ch/résumè

  For a complete description of what makes up a valid URI, see RFC 2396 *Uniform Resource Identifiers (URI): Generic Syntax*.

- The local name uniquely identifies a service or document type within the collection encompassed by a particular namespace. Many webMethods users use a service's unqualified name as its local name. Under this scheme, a service named gl.journals:closeGL would have a local name of closeGL.

Local names follow the same construction rules as NCNames in XML. Basically, a local name can be composed of any combination of letters, digits, or the following symbols:

.	(period)
-	(dash)
_	(underscore)

Additionally, the local name must begin with a letter or an underscore. The following are examples of valid local names:

addCustOrder

authorize_Level1

générent

For specific rules relating to NCNames, see "NCName" definition in the *Namespaces in XML* specification.

Implicit and Explicit Universal Names

Every service or document type that exists on Integration Server has an *explicit* or an *implicit* universal name.

- **An explicit universal name** is a universal name that you specifically assign to a service or document type with Designer. When you assign an explicit universal name, you must specify both the namespace name and the local name.
- **An implicit universal name** is automatically derived from the name of the service or the document type. The implicit name acts as the universal name when a service or document type does not have an explicit universal name. The server derives an implicit name as follows:
  - The *namespace name* is the literal string `http://localhost/` followed by the fully qualified name of the folder in which the service or document type resides on the Integration Server.
  - The *local name* is the unqualified name of the service or document type.

The following table shows the implicit names for a variety of service names:

	The service's implicit universal name is...	
Fully qualified service name	**Namespace name**	**Namespace name**
gl.journals:jrnlEntry	http://localhost/gl.journals	jrnlEntry
gl.journals.query:viewJournals	http://localhost/gl.journals.query	viewJournals
orders:postPO	http://localhost/orders	postPO

Note:

It is possible for an implicit name to match the explicit name of another service. When this condition exists, the explicit name takes precedence. That is, when a universal name is requested, Integration Server searches its registry of explicit names first. If it does not find the requested name there, it looks for a matching implicit name.

Assigning, Editing, or Viewing an Explicit Universal Name

To ensure interoperability with other vendor's implementations of SOAP, Software AG recommends that you always assign explicit universal names to those document types that you want to make available to SOAP clients.

When you assign an explicit universal name, you must enter values in both the **Namespace name** and **Local name** fields. If you specify one field but not the other, you will receive an error message when you attempt to save the service or document type. You will not be permitted to save it until you specify both parts of the universal name.

If you *move* a service or document type, or a folder containing a service or document type, Designer retains the explicit universal name. If you *copy* a service or document type, or a folder containing a service or document type, Designer does not retain the explicit universal name.

Earlier versions of the webMethods SOAP implementation did not include the `http://localhost/` prefix as part of an implicit name. However, the server is backward compatible. It will resolve QNames that clients submit in either the old form (without the http prefix) or the new form (with the http prefix).

To assign, edit, or view a universal name

1. In Package Navigator view, double-click the service or document type whose universal name you want to assign, edit, or view.

2. In the editor, click the service's or document type's title bar to give the service or document type the focus.

3. If you want to assign or edit the universal name, specify the following in the **Universal Name** category of the Properties view:

In this field...	Specify...
Namespace name	The URI that will be used to qualify the name of this service or document type. You must specify a valid absolute URI.
Local name	A name that uniquely identifies the service or document type within the collection encompassed by **Namespace name**. The name can be composed of any combination of letters, digits, or the period (.), dash (-) and underscore (_) characters. Additionally, it must begin with a letter or the underscore character.
	Note:

In this field...	Specify...
	Many webMethods users use the unqualified portion of the service name as the local name.

4. Click **File > Save**.

Deleting an Explicit Universal Name

To delete a universal name

1. In the Package Navigator, open the service or document type whose universal name you want to delete.

2. In the **Universal Name** category of the Properties view, clear the settings in the **Namespace name** and **Local name** fields.

3. Click **File > Save**.

The Universal Name Registry

Integration Server maintains a registry, called the *Universal Name Registry*, which maps explicit universal names to the services and document types that they represent. The registry is generated each time the Integration Server is started and is maintained in memory while the server is running.

When you use the Designer to assign, modify, or delete a universal name, you update the Universal Name Registry. To view the contents of the registry, you can execute the servicepub.universalName:list in Designer and view the contents of the names variable on the Input/Output tab. (This service resides in the WmPublic package.)

Services You Use to Interact with the Universal Name Registry

The following services can be used to display the Universal Name Registry or locate the name of a service or document type associated with an explicit universal name. For more information about these services, see the *webMethods Integration Server Built-In Services Reference*.

Service	Description
pub.universalName:list	Returns a document list containing the entries for the service in the current registry. Each document in the list represents an entry in the registry and contains a service's fully qualified webMethods name and both parts of its explicit universal name.

Service	Description
pub.universalName:listAll	Returns a document list containing the entries for services and document types in the current registry. Each document in the list represents an entry in the registry and contains the fully qualified webMethods name and both parts of its explicit universal name for each service and document type.
pub.universalName:find	Returns the fully qualified service name for a specified explicit universal name.
pub.universalName:findDocumentType	Returns the fully qualified document type name for a specified explicit universal name.

About Service Output Templates

An output template is a web document that is embedded with special codes (tags) that Integration Server processes. These tags instruct Integration Server to perform a specific action and substitute the result of that action in the web document. Typically, you use tags in output templates to insert service output values in web documents returned to clients.

Output templates are used most frequently to customize the HTML page that a service returns to a browser-based application. However, they can also be used to generate an XML document or any other formatted string. For example, you may have a service that retrieves a record from a relational database and uses an output template to format the record as an XML document or a comma-delimited record before returning it to the requester.

Output templates are optional. If a service has an output template assigned to it, the server automatically applies the template to the results of the service (that is, the contents of the pipeline) whenever that service is invoked by an HTTP client. If a service does not have an output template, the server simply returns the results of the service in the body of an HTML document, formatted as a two-column table.

Creating an Output Template

You can use Designer to create an output template, or you can create an output template file using an ordinary text editor.

When you create an output template, keep the following points in mind:

- You must give the output template file a name that is unique within the package in which it resides.
- If you want the template to produce output in XML, WML, or HDML, you must include a <meta> tag in the first line of the template's contents that sets the value of Content Type accordingly (for example, <meta http-equiv="Content-Type" content="text/xml">).
- The default encoding that Integration Server uses to interpret data posted in the resulting browser page is Unicode (UTF-8). If you want to change the encoding to something other than

UTF-8, you must insert a <meta> tag that sets the encoding (for example, <meta http-equiv="Content-Type" charset=iso-8859-1">).

- If you specify a file encoding other than UTF-8 in the <meta> tag of your template's content, the characters that you use in your template (including the data inserted into your template using %VALUE% statements) are limited to those in the character set of the encoding you choose.
- You can reference one output template from within another.

The following procedure describes how to create an output template using Designer.

To create an output template

1. Lock and open the flow service whose output you want to format using an output template.
2. In the Properties view, in the **Name** field under **Output template**, type the name of the file that will contain the output template tags. Alternatively, you can accept the name that Designer suggests.
3. In the **Template source** field, do the following:

   a. Click ....

   b. On the Template Source dialog, type or paste all literal text exactly as you want it to appear in the service output, including HTML, XML, WML, or HDML content as desired. Then, embed any output template tags where you want the server to execute them at run time. For details about template tags, see *Dynamic Server Pages and Output Templates Developer's Guide*.

   c. Click **Save**.

   Designer creates the file in the format FolderName_ServiceName.html and stores the file in the *IntegrationServer_directory*\instances*instance_name*\packages*packageName*\templates directory.

Assigning an Output Template to a Service

Assigning an output template to a service causes the output of the service to be formatted according to the tags the template contains.

When you assign an output template to a service, keep the following points in mind:

- A service can have at most one output template assigned to it at a time.
- You can assign the same output template to more than one service.
- The output template file must reside in the *IntegrationServer_directory*\instances*instance_name*\packages*packageName*\templates directory, where *packageName* is the package in which the service is located.

Note:
If you assign an output template to a service and later copy that service to a different package, you must copy the output template file to the *IntegrationServer_directory*\instances*instance_name*\packages*packageName*\templates directory of the new package. (If you copy an entire package, any output templates will be included automatically.) If the template file has a file extension other than .html, rename the file extension as ".html" so that Designer will recognize its contents.

- The server treats the case of the file name differently depending on which operating system you are using. For example, on a case-insensitive system such as Windows, the server would see the names "template" and "TEMPLATE" as the same name. However, on a case-sensitive system such as UNIX, the server would see these as two different names. If you are trying to assign an existing output template and you enter a file name in the wrong case on a UNIX system, the wrong file name could be assigned as the output template for your service.

To assign an output template to a service

1. Lock and open the flow service whose output you want to format using an output template.
2. In the Properties view, in the **Name** field under **Output template**, type the name of an existing output template file that you want to assign to the service. You do not need to include the path information or the file name extension.
3. If you want to edit the template, do the following:
   a. In the **Template source** field, click ....
   b. On the Template Source dialog, edit the literal text or tags as desired. For details about template tags, see *Dynamic Server Pages and Output Templates Developer's Guide*.
   c. Click **Save**.

   Note:
   Changes you make to an output template affect *all* the services in the package that use the template, not just the service that is currently open in the editor.

Printing a Flow Service

You can use the **View as HTML** command to produce a printable version of a flow service. The resulting HTML page displays all aspects of a flow (its input and output parameters, its flow steps, and pipeline behavior).

Note:
The View as HTML feature is available only for flow services.

To print a flow service

1. In Package Navigator view, right-click the flow service that you want to print and select **View as HTML.**

   Designer creates an HTML page and displays it in your default browser.

2. Use your browser's print command to print the flow.

Comparing Flow Services

You can use the compare tool to compare two flow services on the same server or on different servers. For more information, see "Comparing Integration Server Packages and Elements" on page 921.

10 Building Flow Services

- What Is a Flow Service? 200
- Building Services Using the Tree Tab or Layout Tab 203
- Creating a New Flow Service 203
- Setting Properties for a Flow Step 215
- The INVOKE Step 215
- The BRANCH Step 218
- The REPEAT Step 226
- The SEQUENCE Step 232
- The LOOP Step 234
- The EXIT Step 239
- The MAP Step 241

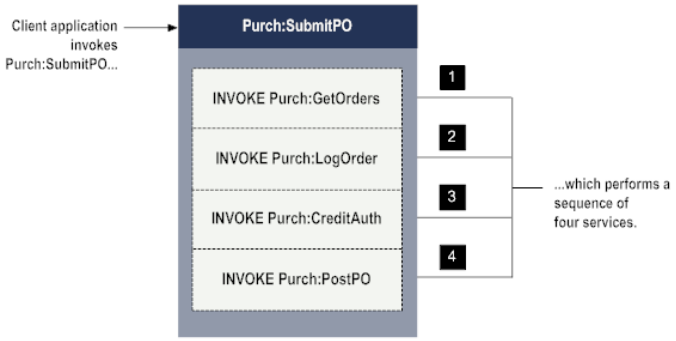
A *flow service* is a service that is written in the webMethods flow language. This simple yet powerful language lets you encapsulate a sequence of services within a single service and manage the flow of data among them.

What Is a Flow Service?

A *flow service* is a service that is written in the webMethods flow language. You can use a flow service to encapsulate a sequence of services within a single service and manage the flow of data among them. For example, you might create a flow service that takes a purchase order from a buyer and executes the following series of services before submitting it to an internal ordering system:

1. Gets a purchase order submitted by a buyer
2. Logs the order in an audit-trail file
3. Performs a credit check
4. Posts the order to the ordering system

Flow services encapsulate other services



Any service can be invoked within a flow (including other flow services). For instance, a flow might invoke a service that you create, any of the built-in services provided with the Integration Server, and/or services from a webMethods add-on product such as the webMethods Adapter for JDBC.

You create flow services using Designer. They are saved in XML files on Integration Server.

Important:
Flow services are written as XML files in a format that is understood by Designer. Create and maintain flow services using Designer. You cannot create or edit a flow service with a text editor.

What Is a Flow Step?

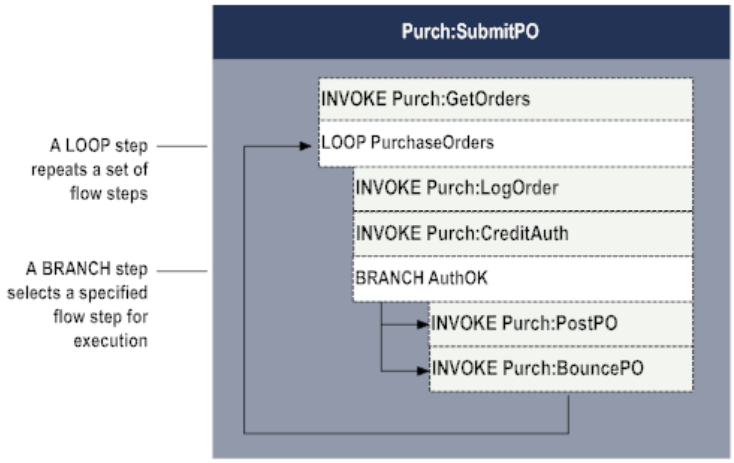
A flow service contains *flow steps*. A flow step is a basic unit of work (expressed in the webMethods flow language) that webMethods Integration Server interprets and executes at run time. The webMethods flow language provides flow steps that invoke services and flow steps that let you edit data in the pipeline.

webMethods flow language also provides a set of *control steps* that allow you to direct the execution of a flow service at run time. The control steps allow you to:

- Conditionally execute a specified sequence based on a field value.
- Retry a specified sequence until it succeeds.
- Repeat a specified sequence (loop) for each element in an array field.

In the following flow service, control steps have been inserted to loop through a subset of the flow service and branch to one of two services in the last step of the loop.

Control steps are used to direct the execution of a flow



A flow service can contain the following types of flow steps:

Invocation Steps	
INVOKE	Executes a specified service. For more information about this step, see "The INVOKE Step" on page 215.
Data-Handling Steps	
MAP	Performs specified editing operations on the pipeline (such as mapping variables in the pipeline, adding variables to the pipeline, and dropping variables from the pipeline). For more information about this step, see "The MAP Step" on page 241.
Control Steps	
BRANCH	Executes a specified flow step based on the value of a specified variable in the pipeline. For more information about this step, see "The BRANCH Step" on page 218.
LOOP	Executes a set of flow steps once for each element in a specified array. For more information about this step, see "The LOOP Step" on page 234.

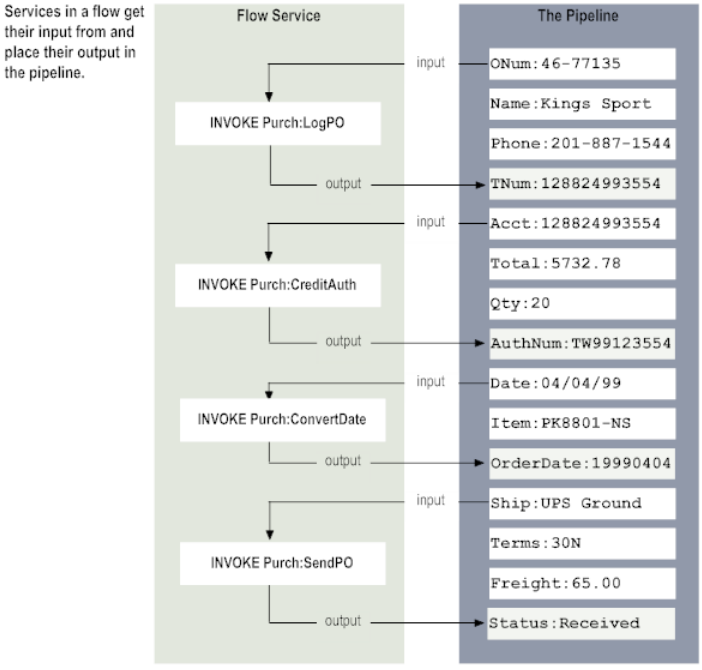
REPEAT	Re-executes a set of flow steps up to a specified number of times based on the successful or non-successful completion of the set. For more information about this step, see "The REPEAT Step" on page 226.
SEQUENCE	Groups a set of flow steps into a series. The SEQUENCE step is implicit in most flow services (that is, the steps in a flow service are treated as a series). However, at times it is necessary to explicitly group a subset of flow steps using SEQUENCE so that they can be treated as a unit. For more information about this flow step, see "The SEQUENCE Step" on page 232.
EXIT	Controls the execution of a flow step (for example, abort an entire flow service from within a series of deeply nested steps, throw an exception without writing a Java service, or exit a LOOP or REPEAT without throwing an exception). For more information about this step, see "The EXIT Step" on page 239.

What Is the Pipeline?

The *pipeline* is the general term used to refer to the data structure in which input and output values are maintained for a flow service. It allows services in the flow to share data.

The pipeline starts with the input to the flow service and collects inputs and outputs from subsequent services in the flow. When a service in the flow executes, it has access to all data in the pipeline at that point.

The pipeline holds the input and output for a flow service



When you build a flow service, you use Designer to specify how information in the pipeline is mapped to and from services in the flow.

Building Services Using the Tree Tab or Layout Tab

In the flow service editor, you can view and build flow services using the Tree tab or Layout tab.

- On the Tree tab, Designer lists flow steps sequentially from top to bottom, and executes steps in that order. The Tree tab provides a more condensed view of a flow service.
- On the Layout tab, a flow service looks similar to a flow chart. Designer displays flow steps from left to right. Lines connect the flow steps and show the order in which the flow steps execute. Designer displays shapes for flow steps as well as for the start and end of the flow service. Steps such as BRANCH, LOOP, and REPEAT that can contain child steps can be collapsed or expanded.

Because the Tree tab and Layout tab provide the same capabilities for building a flow service, work in whichever tab you find easier to use. You can easily switch between the tabs when building a flow service.

Designer uses the Tree tab as the default tab for building and viewing flow services. For this reason, unless specifically stated otherwise, the procedures in the *webMethods Service Development Help* are written for working in the Tree tab in the flow service editor. For information about working in the Layout tab in the flow service editor, see "Working in the Layout Tab" on page 243.

Creating a New Flow Service

You can create a new flow service in the following ways:

- Create an empty flow service and define it yourself by inserting flow steps.
- Create a flow service from a source file, such as an XML Schema, DTD, or XML document. If you are building a flow service that extracts data from an XML document, you can select a source file from which to automatically generate the logic (that is, the set of flow steps) that will get data from a specified document, rather than building this logic manually.

Note:
When you specify a new service name at the time of creating a new REST V2 resource operation, Designerautomatically creates the specified flow service under the same folder as the REST V2 resource. For details, see "Defining a REST V2 Resource Operation" on page 477.

Creating an Empty Flow Service

When you create an empty flow service, you must insert the flow steps manually.

To create an empty flow service

1. In Designer: **File > New > Flow Service**
2. In the New Flow Service dialog box, select the folder in which you want to save the flow service.

3. In the **Element name** field, type a name for the flow service using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

4. If you have a template you want to use to initialize a default set of properties for the service, select if from the **Choose template** list.

5. Click **Next**.

6. On the Select the Source Type panel, select **Empty Flow**.

7. Click **Finish** to create the empty flow service.

8. To insert flow steps, see "Inserting Flow Steps" on page 213.

Creating a Flow Service from an XML Document, DTD, or XML Schema

You can create a flow service based on the structure of a source file, such as an XML Schema definition, DTD, or XML document to automatically generate a service that receives an XML node as input. When creating a flow service this way, Designer generates IS document types, associated schemas, and an XML node based on the source file you provide. Designer adds the XML node as input to the flow service and creates references to the IS document types in the flow service output signature.

Important:
The flow steps produced by this option are no different than those produced by manually inserting INVOKE pub.xml:loadXMLNode and INVOKE pub.xml:queryXMLNode steps in a flow service. After Designer inserts the set of default steps into your flow service, you can edit the default steps and insert additional steps just as you would any ordinary flow service.

Creating a Flow Service from an XML Document

When you create a flow service using the structure of an existing XML document, Designer uses the DTD or XML Schema definition referenced in the XML document you specify to create the elements for the flow service. If the XML document does not reference a DTD or XML Schema definition, Designer creates the document type using the structure of the XML document.

To create a flow service from an XML document

1. In Designer: **File > New > Flow Service**

2. In the New Flow Service dialog box, select the folder in which you want to save the flow service.

3. In the **Element name** field, type a name for the flow service using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

4. If you have a template you want to use to initialize a default set of properties for the service, select if from the **Choose template** list.

5. Click **Next**.

6. On the Select the Source Type panel, select **XML** and click **Next**.

7. On the Select a Source Location panel, under **Source location**, do one of the following in the **File/URL** field:

   - To create the flow service from an XML document that resides on the Internet, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
   - To create the flow service from an XML document on your local file system, type in the path and file name, or click the **Browse** button to navigate to and select the file.

8. Click **Finish** to create the flow service.

Note:
Before running a flow service that expects an XML document as input, you must first create a launch configuration that specifies the XML file, and then debug the service in Designer. For information about creating a launch configuration, see "Creating a Launch Configuration for Running a Service" on page 404.

Creating a Flow Service from a DTD

When you create a flow service from a DTD, Designer uses the elements and attributes defined by the DTD you specify to create the elements for the flow service.

To create a flow service from a DTD

1. In Designer: **File > New > Flow Service**

2. In the New Flow Service dialog box, select the folder in which you want to save the flow service.

3. In the **Element name** field, type a name for the flow service using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

4. Click **Next**.

5. On the Select the Source Type panel, select **DTD** and click **Next**.

6. On the Select a Source Location panel, under **Source location**, do one of the following next to **File/URL**:

   - To create the flow service from an DTD that resides on the Internet, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
   - To create the flow service from a DTD on your local file system, type in the path and file name, or click the **Browse** button to navigate to and select the file.

7. Click **Next**.

8. Under **Select the root node**, select the root element of the DTD.

9. Under **Element reference handling**, select one of the following:

   - Select **Only generate document types for elements with multiple references** to instruct Integration Server to create a separate document type for a referenced element only when the DTD contains multiple references to that element.

     If an element is referenced multiple times, Integration Server creates a separate document type for the element. Integration Server replaces each element reference with a document reference field.

     If an element is referenced only once, Integration Server defines the element in line by replacing the element reference with a document field.

   - Select **Always generate document types for referenced elements** to instruct Integration Server to always create a separate document type for a referenced element even if it is referenced only once. In the document type, Integration Server replaces each element reference with a document reference field.

10. Click **Finish** to create the flow service.

    Integration Server generates the IS document type and IS schema. Designer displays any errors or warnings that occur during document type generation.

> **Note:**
> If the flow service expects an XML document as input, you must create a launch configuration and debug the service in Designer before running it. For more information, see "Creating a Launch Configuration for Running a Service" on page 404.

Creating a Flow Service from an XML Schema Definition

When you create a flow service from an XML Schema definition, Integration Server also creates one or more IS document types and IS schemas.

Keep the following points in mind when creating flow service from an XML Schema definition:

- You can specify whether Integration Server enforces strict, lax, or no content model compliance when generating the document type that is referenced in the flow service. Content models

provide a formal description of the structure and allowed content for a complex type. The type of compliance that you specify can affect whether Integration Server generates an IS document type or flow service from a particular XML Schema definition successfully. Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type or flow service from any XML schema definition that contains those items.

- Integration Server can create separate IS document types for named complex types or expand documents inline within one document type. For more information, see "Determining How to Represent Complex Types in Document Types" on page 543.
- Integration Server can create one field for a substitution group or create fields for every member element in a substitution group. For more information, see "Generating Fields for Substitution Groups" on page 549.
- To create a flow service from an XML Schema definition in CentraSite, Designer must be configured to connect to CentraSite.
- When creating a flow service from an XML Schema definition that contains a large number of complex type definitions, and you want Integration Server to create a separate IS document for each complex type definition, you may need to increase the number of elements that Designer maintains in cache. If the cache is not large enough to include all of the generated IS document types, then Designer will have to repeatedly retrieve the document types from Integration Server while creating the flow service. This increases network traffic and can prolong the time needed to create the flow service. If the cache is large enough to contain all of the IS document types and other elements generated by Designer while creating a flow service, Designer might create the flow service more quickly. To increase the number of elements cached by Designer, see "Caching Elements" on page 72.

To create a flow service from an XML Schema definition

1. In Designer: **File > New > Flow Service**
2. In the New Flow service dialog box, select the folder in which you want to save the flow service.
3. In the **Element name** field, type a name for the flow service using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
4. If you have a template you want to use to initialize a default set of properties for the service, select if from the **Choose template** list.
5. Click **Next**.
6. On the Select a Source Type panel, select **XML Schema**. Click **Next**.
7. On the Select a Source Location panel, under **Source location**, do one of the following to specify the source file for the flow service:

- To use an XML schema definition in CentraSite as the source, select **CentraSite**.
- To use an XML schema definition that resides on the Internet as the source, select **File/URL**. Then, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
- To use an XML Schema definition that resides on your local file system as the source, select **File/URL**. Then, type in the path and file name, or click the **Browse** button to navigate to and select the file.

8. Click **Next**.

9. If you selected CentraSite as the source, under **Select XML Schema from CentraSite**, select the XML Schema definition in CentraSite that you want to use to create the flow service. Click **Next**.

   If Designer is not configured to connect to CentraSite, Designer displays the **CentraSite > Connections** preference page and prompts you to configure a connection to CentraSite.

10. On the Select Processing Options panel, under **Schema domain**, specify the schema domain to which any generated IS schemas will belong. Do one of the following:

    - To add the IS schema to the default schema domain, select **Use default schema domain**.
    - To add the IS schemas to a specified schema domain, select **Use specified schema domain** and provide the name of the schema domain in the text box. A valid schema domain name is any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

11. Under **Content model compliance**, select one of the following to indicate how strictly Integration Server represents content models from the XML Schema definition in the resulting IS document type.

Select...	To...
Strict	Generate the IS document type only if Integration Server can represent the content models defined in the XML Schema definition correctly. Document type generation fails if Integration Server cannot accurately represent the content models in the source XML Schema definition. Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type from any XML schema definition that contains those items.
Lax	When possible, generate an IS document type that correctly represents the content models for the complex types defined in the XML schema definition. If Integration Server cannot correctly represent the content model in the XML Schema definition in the resulting IS document type, Integration Server generates the IS document type using a compliance mode of None.

Select...	To...
	When you select lax compliance, Integration Server will generate the IS document type even if the content models in the XML schema definition cannot be represented correctly.
None	Generate an IS document type that does not necessarily represent or maintain the content models in the source XML Schema definition. When compliance is set to none, Integration Server generates IS document types the same way they were generated in Integration Server releases prior to version 8.2.

12. If you selected strict or lax compliance, next to **Preserve text position**, do one of the following to specify whether document types generated from complex types that allow mixed content will contain multiple **body* fields to preserve the location of text in instance documents.

    - Select the **Preserve text position** check box to indicate that the document type generated for a complex type that allows mixed content preserves the locations for text in instance documents. The resulting document type contains a **body* field after each field and includes a leading **body* field. In instance documents for this document type, Integration Server places text that appears after a field in the **body*.
    - Clear the **Preserve text position** check box to indicate that the document type generated for a complex type that allows mixed content does not preserve the locations for text in instance documents. The resulting document type contains a single **body* field at the top of the document type. In instance documents for this document type, text data around fields is all placed in the same **body* field.

13. If this document type will be used as the input or output signature of a service exposed as a web service and you want to enable streaming of MTOM attachment for elements of type base64Binary, select the **Enable MTOM streaming for elements of type base64Binary** check box.

    For more information about streaming of MTOM attachments, see "Working with Web Services" on page 743.

14. If you want Integration Server to use the Xerces Java parser to validate the XML Schema definition, select the **Validate schema using Xerces** check box.

    Note: Integration Server automatically uses an internal schema parser to validate the XML Schema definition. However, the Xerces Java parser provides stricter validation than the Integration Server internal schema parser. As a result, some schemas that the internal schema parser considers to be valid might be considered invalid by the Xerces Java parser.

15. Click **Next**.

16. On the Select Root Node panel, under **Select the root node,** select the elements that you want to use as the root elements for the IS document type. The resulting IS document type will contain all of the selected root elements as top-level fields in the generated IS document type

    To select multiple elements, press the CTRL key while selecting elements.

    If Integration Server determines that the XML Schema definition is invalid, the Select Root Node panel displays an error message to that effect. Click **Cancel** to abandon the attempt to create a document type.

17. Under **Element reference handling**, select one of the following to determine how Integration Server handles references to global elements of complex type:

Select...	To...
Only generate document types for elements with multiple references	Instruct Integration Server to create a separate document type for a referenced element only when the XML Schema definition contains multiple references to that element. If an element is referenced multiple times, Integration Server creates a separate document type for the element. Integration Server replaces each element reference with a document reference field. If an element is referenced only once, Integration Server defines the element in line by replacing the element reference with a document field.
Always generate document types for referenced elements	Instruct Integration Server to always create a separate document type for a referenced element even if it is referenced only once. In the document type, Integration Server replaces each element reference with a document reference field

Note: Integration Server always replaces an element reference to an element declaration of simple type with an inline field of type String.

18. Under **Complex type handling**, select one of the following to indicate how Integration Server handles references to named complex type definitions:

Select...	To...
Expand complex types inline	Use a document field defined in line to represent the content of a referenced complex type definition.
Generate document types for complex types	Create a separate IS document type to represent the content for a referenced complex type definition. The resulting IS document type for the root element represents the element

Select...	To...
	of complex type using a document reference field. In turn, this document reference field refers to the IS document type created for the complex type definition. Integration Server generates a separate IS document type for any types derived from the referenced complex types. For more information about derived types, see "Derived Types and IS Document Types" on page 545.

Note: Integration Server always represents an anonymous complex type using a document field defined inline.

19. If you selected **Generate document types for complex types** and you want to register each document type with the complex type definition from which it was created, select the **Register document type with schema type** check box.

Note:
If you want derived type support for document creation and validation, select the **Register document types with schema type** check box. For more information, see "Registering Document Types with Their Schema Types" on page 547.

20. If you want Integration Server to generate IS document types for all complex types in the XML Schema definition regardless of whether the types are referenced by elements or other type definitions, select the **Generate document types for all complex types in XML Schema** check box.

    If you leave this check box cleared, Integration Server generates a separate IS document type for a complex type only if the complex type is referenced or is derived from a referenced complex type.

21. If any of the root elements you selected for the IS document type contain a namespace URI and you want to create a new namespace prefix for it, click **Next**. Otherwise, continue with step 22.

22. On the Assign Prefixes panel, if you want the IS document type to use different prefixes than those specified in the XML schema definition, select the prefix you want to change and enter a new prefix. Repeat this step for each namespace prefix that you want to change.

Note:
The prefix you assign must be unique and must be a valid XML NCName as defined by the specification http://www.w3.org/TR/REC-xml-names/#NT-NCName.

23. Click **Finish**.

    Integration Server generates the flow service, IS document type(s), and IS schema and saves it on the server. Designer displays them in the Package Navigator view.

Notes:

- Integration Server uses the internal schema parser to validate an XML schema definition. If you selected the **Validate schema using Xerces** check box, Integration Server also uses the Xerces Java parser to validate the XML Schema definition. With either parser, if the XML Schema does not conform syntactically to the schema for XML Schemas defined in *XML Schema Part 1: Structures* (which is located at http://www.w3.org/TR/xmlschema-1), Integration Server does not create an IS schema. Instead, Designer displays an error message that lists the number, title, location, and description of the validation errors within the XML Schema definition. If only warnings occur, Designer generates the IS schema.

  Note: Integration Server uses Xerces Java parser version J-2.11.0. Limitations for this version are listed at http://xerces.apache.org/xerces2-j/xml-schema.html.

- When validating XML schema definitions, Integration Server uses the Perl5 regular expression compiler instead of the XML regular expression syntax defined by the World Wide Web Consortium for the XML Schema standard. As a result, in XML schema definitions consumed by Integration Server, the pattern constraining facet must use valid Perl regular expression syntax. If the supplied pattern does not use proper Perl regular expression syntax, Integration Server considers the pattern to be invalid.

  Note:
  If the watt.core.datatype.usejavaregex configuration parameter is set to true, Integration Server uses the Java regular expression compiler instead of the Perl5 regular expression compiler. When the parameter is true, the pattern constraining facet in XML schema definitions must use valid syntax as defined by the Java regular expression.

- If you selected strict compliance and Integration Server cannot represent the content model in the complex type accurately, Integration Server does not generate any IS document types for the flow service.

- If you selected lax compliance and indicated that Integration Server should preserve text locations for content types that allow mixed content (you selected the **Preserve text position** check box), Integration Server adds **body* fields in the document type only if the complex type allows mixed content and Integration Server can correctly represent the content model declared in the complex type definition. If Integration Server cannot represent the content model in an IS document type, Integration Server adds a single **body* field to the document type.

- The contents of an IS document type with a **Model type** property value other than "Unordered" cannot be modified.

- If the XML schema definition contains an element reference to an element declaration whose type is a named complex type definition (as opposed to an anonymous complex type definition), Integration Server creates an IS document type for the named complex type definition. In the IS document type for the root element, Integration Server uses document reference field to represent the element reference. An exception to this behavior is the situation in which the element reference is the only reference to the complex type definition and the **Only generate document types for elements with multiple references** option is selected. In this situation, Integration Server uses document field defined in line to represent the content of the referenced complex type.

- Integration Server uses the prefixes declared in the XML Schema or the ones you specified as part of the field names. Field names have the format *prefix*:*elementName* or *prefix*:*@attributeName*.
- If the XML Schema does not use prefixes, the Integration Server creates prefixes for each unique namespace and uses those prefixes in the field names. Integration Server uses “ns” as the prefix. The first namespace is “ns1” and the second namespace is “ns2”.
- When creating a flow service from an XML Schema definition that imports multiple schemas from the same target namespace, Integration Server throws Xerces validation errors indicating that the element declaration, attribute declaration, or type definition cannot be found. The Xerces Java parser honors the first <import> and ignores the others. To work around this issue, you can do one of the following:
  - Combine the schemas from the same target namespace into a single XML Schema definition. Then change the XML schema definition to import the merged schema only.
  - When creating the flow service, clear the **Validate schema using Xerces** check box to disable schema validation by the Xerces Java parser. When generating the flow service. Integration Server will not use the Xerces Java parser to validate the schemas associated with the XML Schema definition.
- Before running a flow service that expects an XML document as input, you must first create a launch configuration that specifies the XML file, and then debug the service in Designer. For information about creating a launch configuration, see “Creating a Launch Configuration for Running a Service” on page 404.

Inserting Flow Steps

Flow steps call previously built services and direct the flow of data within a flow service. You can use the button on flow service toolbar or the Palette view to insert flow steps into a flow service.

The Palette view is located at the right of the flow service editor. Click to show the Palette view. Click to hide the Palette view.

To insert a flow step

1. In the Package Navigator view, open the flow service in which you want to insert flow steps.
2. Do one of the following:
   - Click the button next to on the flow service editor toolbar and select the flow step that you want to insert.
   - In the Palette view, select the flow step that you want to insert and drag it to the flow service editor.

Changing the Position of a Flow Step

Integration Server executes flow steps in the order in which they appear in the editor.

To move a step up or down in a flow service

1. On the Tree tab, select the flow step that you want to move.
2. Use the following toolbar buttons to move the step.

To...	Click this button...
Move the flow step up in the list	
Move the flow step down in the list	

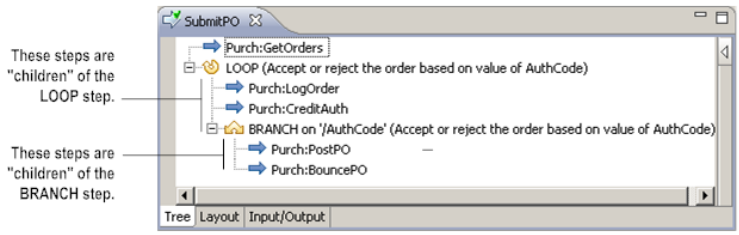
Tip:
You can also move a flow step by dragging it up or down with your mouse.

Changing the Level of a Flow Step

Some flow steps have subordinate steps on which they operate. Subordinate steps are referred to as *children*. For example, when you use the LOOP step, the set of steps that make up the loop are referred to as children of that LOOP step.

Children are specified by indenting them beneath their parent flow step. In the following example, the top step has three children. Note that one of its children is a BRANCH step, which has its own set of children.

Child steps are indented beneath their parent step



To promote or demote a flow step within a parent/child hierarchy

1. On the Tree tab, select the flow step that you want to move.
2. Use the following toolbar buttons to move the step left or right beneath the current parent step.

To...	Click this button...
Demote a flow step in the hierarchy (that is, make the selected step a child of the preceding parent step)	

To...	Click this button...
This button will only be available if you select a step that can become a child.	
Promote a flow step in the hierarchy (that is, move the step one level up in the hierarchy)	

Setting Properties for a Flow Step

Every flow step is associated with a unique set of properties. The properties for a flow step are displayed in the Properties view. Values that you specify in the Properties view apply only to the selected step in the editor.

Although each type of flow step has a set of unique properties, they all have the following properties:

Property	Description
Comments	Assigns an optional descriptive comment to the selected flow step.
Label	Assigns a name to the selected flow step. When a label is assigned, that label appears next to the step in the editor. The label allows you to reference that flow step in other flow steps. In addition, you use the label to control the behavior of certain flow steps. For example, the BRANCH step uses the **Label** property to determine which alternative it is supposed to execute. See "The BRANCH Step" on page 218 and "The EXIT Step" on page 239 for additional information about this use of the label property.

The INVOKE Step

Use the INVOKE step to request a service within a flow. You can use the INVOKE step to:

- Invoke *any* type of service, including other flow services and web service connectors.
- Invoke any service for which the caller of the current flow has access rights on the local webMethods Integration Server.
- Invoke built-in services and services on other webMethods Integration Servers.
- Invoke flow services recursively (that is, a flow service that calls itself). If you use a flow service recursively, bear in mind that you must provide a means to end the recursion.
- Invoke any service, validating its input and/or output.

Specifying the Service Property

The INVOKE step's **Service** property specifies which service will be invoked at run time. When you insert an INVOKE step, Designer automatically assigns the name of that service to the **Service** property.

If you want to change the service assigned to an INVOKE step, you edit the **Service** property. You edit this property in one of two ways:

- By clicking **Service**. Service property's edit button (...) and selecting a service from the Select dialog box. This is the preferred method.
- By typing the name of a service in the **Service** text box. When you specify a service in this manner, keep the following points in mind:
  - You must specify the service's fully qualified name in *folderName*:*serviceName* format.

    Example `purchasing.orders:getOrders`
  - You must specify the service's name *exactly* as it is defined on the server. Service names are case sensitive.

Invoking a Built-In Service

There is an extensive set of built-in services that you can invoke from a flow service. The webMethods library includes services for doing such things as transforming data values, performing simple mathematical operations, extracting information from XML documents, and accessing databases.

Built-in services reside in the WmPublic package. For a complete description of these services, see the *webMethods Integration Server Built-In Services Reference*.

Note:
If you are using any adapters (for example, the webMethods Adapter for JDBC), you will have additional built-in services, which are provided by the adapters. See the documentation provided with those adapters for details.

Invoking a Service on Another Integration Server

You can use the built-in service pub.remote:invoke to invoke a service on a remote Integration Server and return the results. The remote server is identified by an alias, which is configured on the **Remote Servers** screen in the Integration Server Administrator. The pub.remote:invoke service automatically handles opening a session and authentication on the remote server.

The pub.remote:invoke service resides in the WmPublic package and requires the alias of the remote server and the fully qualified name of the service that you want to invoke as input. For a complete description of this service, see the *webMethods Integration Server Built-In Services Reference*.

Building an Invoke Step

Use the following procedure to invoke a service within a flow service. For information about how to can specify input/output validation for the service, see "Specifying Input/Output Validation via the INVOKE Step" on page 296.

To build an INVOKE step

1. Open the flow service in which you want to invoke another service. In the editor, select the step immediately above where you want to insert the INVOKE step.

2. Do one of the following:

   - Click the ▾ button next to ⇨ on the flow service editor toolbar and click ➡. In the **Open** dialog box, navigate to the service you want to invoke.
   - Click ◁ by the side of the flow service editor to open the Palette view and select the flow step that you want to insert and drag it to the flow service editor.
   - Select one or more services in Package Navigator view and drag them to the desired position within the flow in the editor. The services must reside on the same server as the flow service.

3. Complete the following fields in the Properties view:

For this property...	Specify...
Service	The fully qualified name of the service that will be invoked at run time. When you insert a service, Designer automatically assign the name of that service to the **Service** property. If you want to change the service that is invoked, specify the service's fully qualified name in the format *folderName*:*serviceName* or click … and select a service from the list.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the

For this property...	Specify...
	watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Validate input	Whether or not you want the server to validate the input to the service against the service input signature. Select **True** to validate the input. Select **False** if you do not want to validate the input.
Validate output	Whether or not you want the server to validate the output of the service against the service output signature. Select **True** to validate the output. Select **False** if you do not want to validate the output.

4. If necessary, on the **Pipeline** view, link **Pipeline In** variables to **Service In** variables. Link **Service Out** variables to **Pipeline Out** variables. For more information about linking variables to a service, see "About Linking Variables" on page 256.

5. Click **File > Save**.

Tip:
In Designer, clicking the ▾ button next to ⇨ or opening the Palette view displays a list of commonly used services. You can edit the **Window > Preferences > Software AG > Service Development > Flow Service Editor** preferences to customize this list of services to suit your needs.

The BRANCH Step

The BRANCH step allows you to conditionally execute a step based on the value of a variable at run time. For example, you might use a BRANCH step to process a purchase order one way if the *PaymentType* value is "CREDIT CARD" and another way if it is "CORP ACCT".

When you build a BRANCH step, you can:

- **Branch on a switch value.**Use a variable to determine which child step executes. At run time, the BRANCH step matches the value of the switch variable to the **Label** property of each of its targets. It executes the child step whose label matches the value of the switch.
- **Branch on an expression.** Use an expression to determine which child step executes. At run time, the BRANCH step evaluates the expression in the **Label** property of each child step. It executes the first child step whose expression evaluates to "true."

Important:
You cannot branch on a switch value *and* an expression for the same BRANCH step. If you want to branch on the value of a single variable and you know the possible run-time values of the switch variable *exactly*, branch on the switch value. If you want to branch on the values of more than one variable or on a range of values, branch on expressions.

Branching on a Switch Value

When you branch on a switch value, you branch on the value of a single variable in the pipeline.

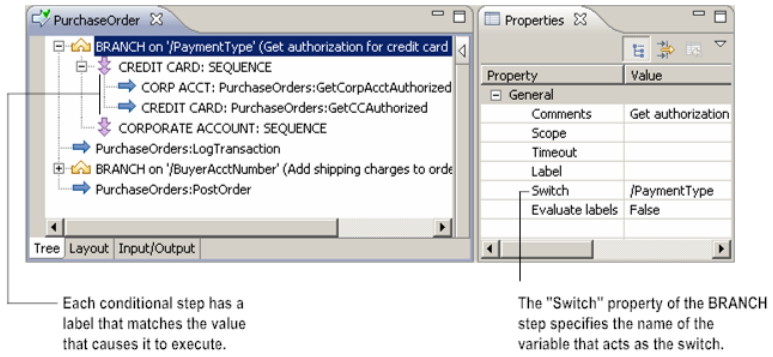
To branch on a switch value

1. Create a list of the conditional steps (target steps) and make them children of the BRANCH step.

2. In the Properties view for the BRANCH step, specify in the **Switch** property the name of the pipeline variable whose value will act as the switch. For more information about this property, see "Specifying the Switch Value" on page 219.

3. In the **Label** property of each target step, specify the value that will cause that step to execute. For more information about this property, see "Specifying the Label Value" on page 219.

Simple BRANCH step that branches on a switch value



Specifying the Switch Value

The variable you use as the switch variable:

- Must be a String or constrained Object variable.
- Must be a variable that can exist in the pipeline when the BRANCH step is executed at run time.
- Must be formatted as *document/documentVariable*, if you are specifying a field in a document as the switch variable (for example, *BuyerInfo/AccountNum*).

Specifying the Label Value

At run time, the BRANCH step compares the value of the switch variable to the **Label** property of each of its targets. It executes the target step whose label matches the value of the switch variable.

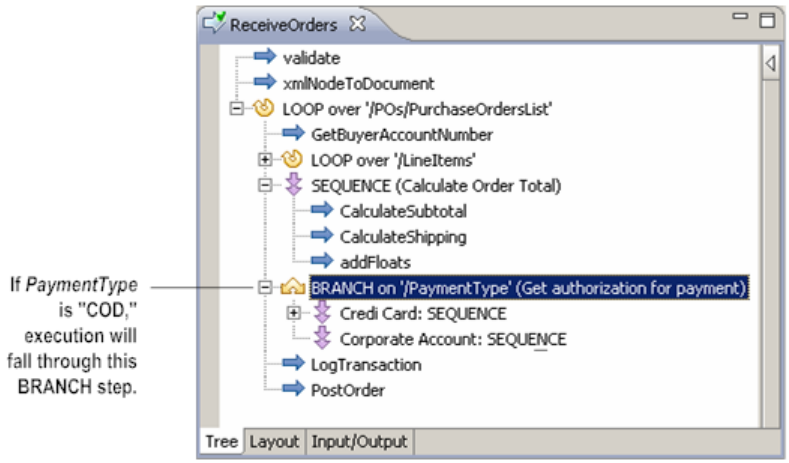
You can use a regular expression to specify the matching value for a BRANCH step. To do so, use the following syntax to specify the value in **Label**:

/RegularExpression/

For example, if you want to select a step based on whether a PO number starts with the string "REL" you use `/^REL/` as the value of **Label**. For more information, see "Regular Expressions" on page 1129.

Unlike other flow steps whose children execute in sequence at run time, only *one* child of a BRANCH step is executed: the target whose label matches the value of the switch variable. If none of the targets match the switch variable, none of them are performed, and execution "falls through" to the next step in the flow service. For example, in the following flow service, execution passes directly to the LogTransaction service if the value of *PaymentType* is "COD."

An unmatched value will fall though the BRANCH



Keep the following points in mind when assigning labels to the targets of the BRANCH step:

- You must give each target step a label unless you want to match an empty string. For that case, you leave the **Label** property blank. For more about matching an empty string, see "Branching on Null and Empty Values" on page 221.
- Each **Label** value must be unique within the BRANCH step.
- When you specify a literal value as the **Label** of a child step, the value you specify must match the run-time value of the switch variable *exactly*. The **Label** property is case sensitive.
- You can use a regular expression as the value of **Label** instead of a literal value.
- You can match a null value by using the **$null** value in the **Label** property. For more information about specifying a null value, see "Branching on Null and Empty Values" on page 221.
- You can designate a default step for all unmatched cases by using the **$default** value in the **Label** property. For more information about using the **$default** setting, "Specifying a Default Step" on page 223.

Branching on an Expression

When you branch on an expression, you assign an expression to each child of a branch step. At run time, the BRANCH step evaluates the expressions assigned to the child steps. It executes the first child step with an expression that evaluates to true.

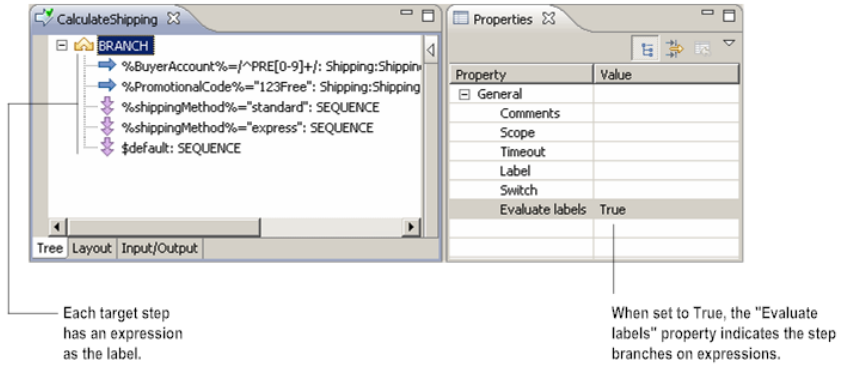
To branch on an expression

1. Create a list of the conditional steps (target steps) and make them children of the BRANCH step.

2. In the Properties view for the BRANCH step, set **Evaluate labels** to **True**.

3. In the **Label** property of each target, specify the expression that, when true, will cause the target step to execute. The expressions you create can include multiple variables and can specify a range of values for variables. Use the syntax provided by webMethods to create the expression. For more information about expression syntax, see "Conditional Expressions" on page 1113.

Simple BRANCH step that branches on an expression



Keep in mind that only one child of a BRANCH step is executed: the first target step whose label contains an expression that evaluates to true. If none of the expressions evaluate to true, none of the child steps are invoked, and execution falls through to the next step in the flow service. You can use the **$default** value in the **Label** property to designate a default step for cases where no expressions evaluate to true. For more information about using the **$default** value, see "Specifying a Default Step" on page 223.

Important:
The expressions you create for the children of a BRANCH step need to be mutually exclusive (only one condition should evaluate to true at run time).

Branching on Null and Empty Values

When you build a BRANCH step, you can include target steps that match null or empty switch values. The BRANCH step considers a switch value to be *null* if the variable does not exist in the

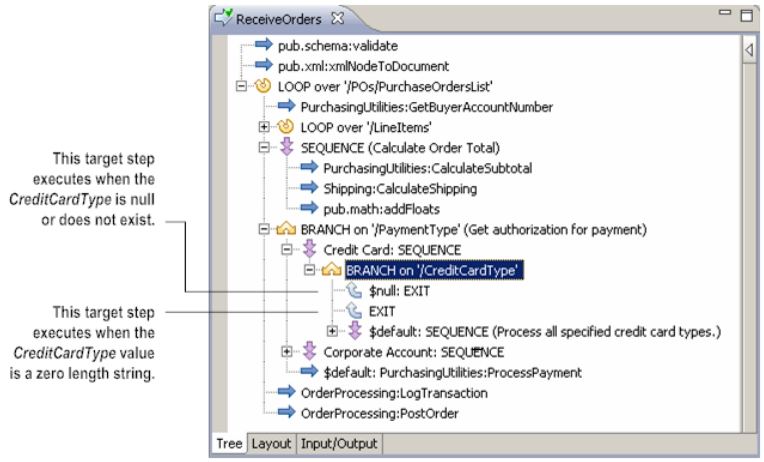
pipeline or is explicitly set to null. The BRANCH step considers a switch value to be an *empty* string if the variable exists in the pipeline but its value is a zero length string. To branch on null or empty values, set the **Label** property for the target step as follows.

To BRANCH on...	Do the following...
A null value	Set the **Label** property to **$null**. At run time, the BRANCH step executes the target step with the **$null** label if the switch variable is explicitly set to null or does not exist in the pipeline. You can use **$null** with any type of switch variable.
An empty string	Leave the **Label** property blank (empty). At run time, the BRANCH step executes the target step with no label if the switch variable is present, but contains no characters. You can use an empty value only when the switch variable is of type String.

Important:
If you branch on expressions (**Evaluate labels** is set to **True**), you cannot branch on null or empty values. When executing the BRANCH step and evaluating labels, Integration Server ignores target steps with a blank or **$null** label.

The following example shows a BRANCH step used to authorize a credit card number based on the buyer's credit card type (*CreditCardType*). It contains three target steps. The first target step handles situations where the value of *CreditCardType* is null or where *CreditCardType* does not exist in the pipeline. The second target step handles cases where the value of *CreditCardType* is an empty string. (Note that the first two target steps are EXIT steps that will return a failure condition when executed.) The third target step has the **$default** label, and will process all specified credit card types.

BRANCH that contains target steps to match null values or empty strings

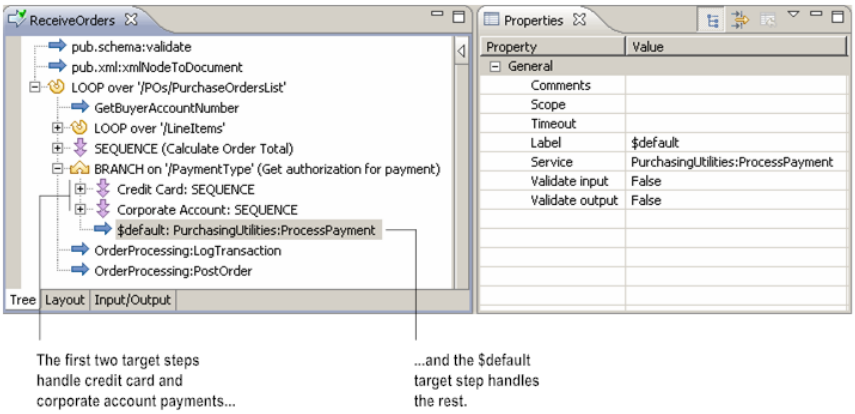


Specifying a Default Step

If you want to prevent the service from falling through a BRANCH step when an unmatched value occurs at run time, include a *default* target step to handle unmatched cases. To specify the default alternative of a BRANCH step, set the **Label** property to **$default**.

The following example shows a BRANCH step that is used to authenticate payment for an order based on the type of payment (*PaymentType*). It contains three target steps. The first target step handles orders paid for by Credit Card. The second target step handles orders paid for through a Corporate Account. The third target step has the **$default** label and will process all other payment types.

The default step is set to $default



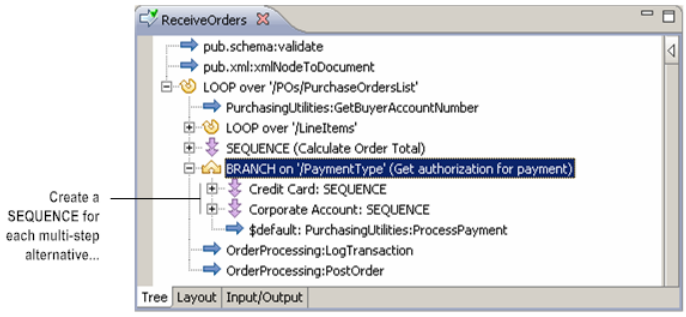
Important:
You can only have one default target step for a BRANCH step. Designer always evaluates the default step last. The default step does not need to be the last child of the BRANCH step.

Using a SEQUENCE as the Target of a BRANCH

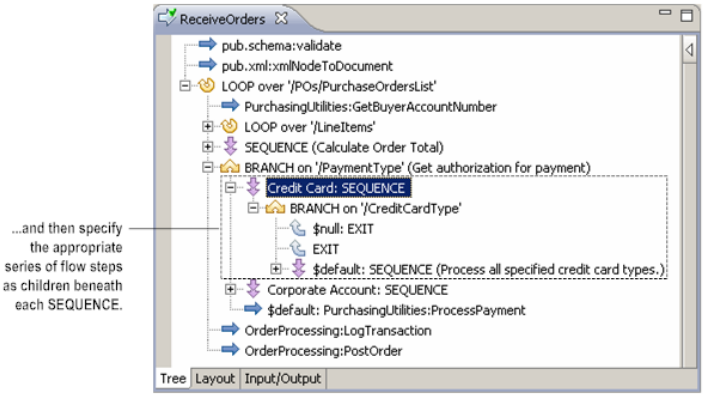
In many cases, you may want a BRANCH step to conditionally execute a series of multiple steps rather than just a single step. For these cases, you can use the SEQUENCE step as the target step and group a series of flow steps beneath it.

The following example illustrates a service that accepts a purchase order and processes it one of three ways depending on the payment type specified in the *PaymentType* variable. Because a series of steps are needed to process the PO in each case, the targets of the BRANCH are defined as SEQUENCE steps, and the appropriate series of steps are specified as children beneath each SEQUENCE.

Use a SEQUENCE step as the target for a multi-step alternative



Define a multi-step alternative in a SEQUENCE



The SEQUENCE step that you use as a target for a BRANCH can contain any valid flow step, including additional BRANCH steps. For additional information about building a SEQUENCE, see "The SEQUENCE Step" on page 232.

Building a BRANCH Step

Use the following procedure to build a BRANCH step in a flow service.

To build a BRANCH step

1. If you are inserting a BRANCH step into an existing flow service, display that service in the editor and highlight the step immediately above where you want the BRANCH step inserted.

2. Do one of the following:

   - Click the ▾ button next to ⇨ on the flow service editor toolbar and click ⌂.
   - Click ◁ by the side of the flow service editor to open the Palette view. Click ⌂ and drag it to the flow service editor.

3. Complete the following fields on the Properties view:

For this property...	Specify...
Comments	An optional descriptive comment for this step.
Scope	The name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Label	An optional name for this specific step, or a null, unmatched, or empty string (**$null**, **$default**, blank). For more information about branching on null or empty values, see "Branching on Null and Empty Values" on page 221. If you use this step as a target for another BRANCH or an EXIT step, you *must* specify a value in the **Label** property. For more information about the EXIT step, see "The EXIT Step" on page 239.
Switch	The name of the String or constrained Object variable whose value will be used to determine which child step to execute at run time. Do not specify a switch variable if you set the **Evaluate labels** property to **True**.
Evaluate label	Whether or not you want to evaluate labels of child steps as conditional expressions. Select **True** to branch on expressions. Select **False** (the default) if you want to branch on the switch value.

4. Insert the conditional steps that belong to the BRANCH (that is, its children) using the following steps:

   a. Insert a flow step by clicking the ▾ button next to on the flow service editor toolbar and clicking the required flow step.

   b. Indent the flow step using on the flow service editor toolbar to make it a child of the BRANCH step.

c. In the **Label** property on the Properties view, specify the switch value that will cause this step to execute at run time.

To match...	Specify...
That exact string	A string
The String representation of the object's value **Example for Boolean object** `true` **Example for Integer object** `123`	A constrained object value
Any string fitting the criteria specified by the regular expression **Example** `/^REL/`	A regular expression
An empty string	A blank field
A null value	**$null**
Any unmatched value (that is, execute the step if the value does not match any other label)	**$default**

d. Set other properties as needed.

Important:
If you are branching on expressions, make sure the expressions you assign to the target steps are mutually exclusive. In addition, do not use null or empty values as labels when branching on expressions. The BRANCH step ignores target steps with a **$null** label or blank label.

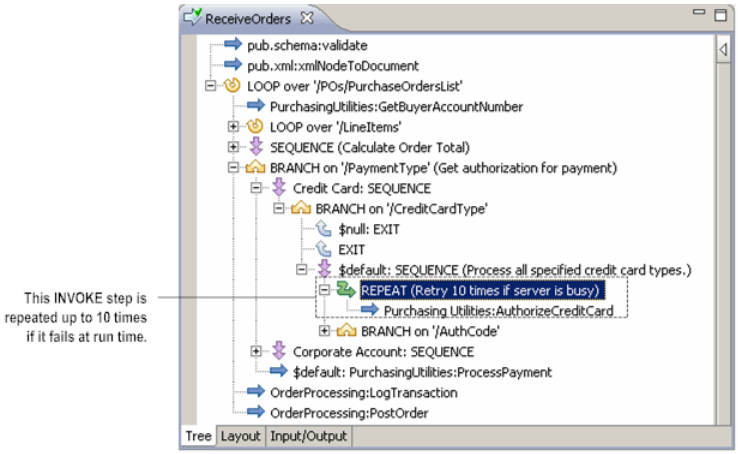
5. Click **File > Save**.

The REPEAT Step

The REPEAT step allows you to conditionally repeat a sequence of child steps based on the success or failure of those steps. You can use REPEAT to:

- **Re-execute (retry) a set of steps if any step within the set fails.** This option is useful to accommodate transient failures that might occur when accessing an external system (for example, databases, ERP systems, web servers, or web services) or device.
- **Re-execute a set of steps until one of the steps within the set fails.**This option is useful for repeating a process as long as a particular set of circumstances exists (for example, data items exist in a data set).

Use REPEAT to re-execute one or more steps



Specifying the REPEAT Condition

When you build a REPEAT step, you set the **Repeat on** property to specify the condition (success or failure) that will cause its children to re-execute at run time.

If you set "Repeat on" to…	The REPEAT step…
FAILURE	Re-executes the set of child steps if any step in the set fails.
SUCCESS	Re-executes the set of child steps if all steps in the set complete successfully.

Setting the REPEAT Counter

The REPEAT step's **Count** property specifies the maximum number of times the server re-executes the child steps in the REPEAT step.

If you set "Count" to…	The REPEAT step…
0	Does not re-execute children.
Any value > 0	Re-executes children up to this number of times.
-1 or blank	Re-executes children as long as the specified **Repeat on** condition is true.

Important:
Note that children of a REPEAT *always* execute at least once. The **Count** property specifies the maximum number of times the children can be *re-executed*. At the end of an iteration, the server checks to see whether the condition (that is, failure or success) for repeating is satisfied. If the condition is true and the **Count** is not met, the children are executed again. This process continues until the repeat condition is false or **Count** is met, whichever occurs first. (In other words, the

maximum number of times that children of a REPEAT will execute when **Count** is > -1, is **Count**+1.)

When Does REPEAT Fail?

The following conditions cause the REPEAT step to fail:

If "Repeat on" is set to…	The REPEAT step fails if…
SUCCESS	A child within the REPEAT block fails.
FAILURE	The **Count** limit is reached before its children execute successfully.

If the REPEAT step is a child of another flow step, the failure is propagated to its parent.

Using REPEAT to Retry a Failed Step

If your flow invokes services that access external systems, you can use the REPEAT step to accommodate network errors, such as busy servers or connection errors, at run time. If you use the REPEAT step for this purpose, keep the following points in mind:

- The following types of failures satisfy the FAILURE condition:
  - Expiration of a child step's **Timeout** limit
  - An exception thrown by a Java service
  - A document query that returns an unpermitted null value
- If you specify multiple children under a REPEAT step, the failure of any one of the children will cause the entire set of children to be re-executed.
- The REPEAT step immediately exits a set of children at the point of failure for the current iteration. That is, if the second child in a set of three fails, the third child is not executed in that iteration of the REPEAT step.
- When **Repeat on** is set to FAILURE, the failure of a child within a REPEAT step does not cause the REPEAT step itself to fail unless the **Count** limit is also reached.
- When **Repeat on** is set to FAILURE and an iteration fails, the pipeline is restored to its original state as it was before that iteration started. Rollback operations are performed on the first level of the pipeline only. That is, first-level variables are restored to their original values before the iteration failed, but the server does not roll back changes to any documents to which the first-level variables refer.
- The **Timeout** property for the REPEAT step specifies the amount of time in which the entire REPEAT step, including all of its possible iterations, must complete. When you use REPEAT to retry on failure, you may want to leave the **Timeout** value at 0 (no limit) or set it to a very high value. You can also set the property to the value of a pipeline variable by typing the name of the variable between % symbols. The variable you specify must be a String.

- As a developer, you must be thoroughly familiar with the processes you include within a REPEAT step. Make certain that the child steps you specify can safely be repeated in the event that a failure occurs. You don't want to use REPEAT if there is the possibility that a single action, such as accepting an order or crediting an account balance, could be applied twice.

To build a REPEAT step that re-executes failed steps

1. If you are inserting a REPEAT step into an existing flow service, display that service in the editor and highlight the step immediately above where you want the REPEAT step inserted.

2. Do one of the following:

   - Click the ▾ button next to ⇨ on the flow service editor toolbar and click .
   - Click ◁ by the side of the flow service editor to open the Palette view. Click and drag it to the flow service editor.

3. Complete the following fields on the Properties view:

For this property...	Specify...
Comments	An optional descriptive comment for this step.
Scope	The name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Label	An optional name for this specific REPEAT step, or a null, unmatched, or empty string (**$null**, **$default**, blank). **Important:** If you use this step as a target for a BRANCH or EXIT step, you must specify a value in the **Label** property. For more information about

For this property...	Specify...
	the BRANCH and EXIT steps, see "The BRANCH Step" on page 218 or "The EXIT Step" on page 239.
Count	The maximum number of times you want the children to be re-executed. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols (for example, `%servicecount%`). The variable you specify must be a String. If you want the children re-executed until they are all successful (that is, no maximum limit), set this value to –1.
Repeat interval	The length of time (in seconds) that you want the server to wait between iterations of the children. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols (for example, `%waittime%`). The variable you specify must be a String.
Repeat on	**FAILURE**

4. Beneath the REPEAT step, use the following steps to insert each step that you want to repeat:

   a. Insert a flow step using the buttons on the flow service editor toolbar.

   b. Indent that flow step using on the flow service editor toolbar. (Make it a child of the REPEAT step.)

   c. Set the properties for the child step as needed.

5. Click **File > Save**.

Using REPEAT to Retry a Successful Step

Apart from using REPEAT to retry a failed step, you can also use it as a looping device to repeat a series of steps until a failure occurs.

If you use the REPEAT step to re-execute successful child steps, keep the following points in mind:

- The success condition is met if all children of the REPEAT step execute without returning a single exception.
- If one child in the set fails, the REPEAT step exits at the point of failure, leaving the remaining children unexecuted.

To build a REPEAT step that repeats a set of successful steps

1. If you are inserting a REPEAT step into an existing flow service, display that service in the editor and highlight the step immediately above where you want the REPEAT step inserted.

2. Do one of the following:

   - Click the ▾ button next to ⇨ on the flow service editor toolbar and click ⇨.
   - Click ◁ by the side of the flow service editor to open the Palette view. Click ⇨ and drag it to the flow service editor.

For this property...	Specify...
Comments	An optional descriptive comment for this step.
Scope	The name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Label	An optional name for this specific step, or a null, unmatched, or empty string (**$null**, **$default**, blank). **Important:** If you use this step as a target for a BRANCH or EXIT step, you must specify a value in the label property. For more information about the BRANCH and EXIT steps, see "The BRANCH Step" on page 218 or "The EXIT Step" on page 239.
Count	The maximum number of times you want the children to be re-executed. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols (for example, `%servicecount%`). The variable you specify must be a String. If you want the children re-executed until any one of them fails (that is, no maximum limit), set this value to –1.

For this property...	Specify...
Repeat interval	The length of time (in seconds) that you want the server to wait between iterations of the children. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols (for example, `%waittime%`). The variable you specify must be a String.
Repeat on	**SUCCESS**

4. Beneath the REPEAT step, use the following steps to insert each step that you want repeat:

   a. Insert a flow step using the buttons on the flow service editor toolbar.

   b. Indent that flow step using on the editor toolbar to make it a child of the REPEAT step.

   c. Set the properties for the child step as needed.

5. Click **File > Save**.

The SEQUENCE Step

You use the SEQUENCE step to build a set of steps that you want to treat as a group. Steps in a group are executed in order, one after another. By default, all steps in a flow service, except for children of a BRANCH step, are executed as though they were members of an implicit SEQUENCE step. That is, the flow steps execute in order, one after another. However, there are times when it is useful to explicitly group a set of flow steps. The most common reasons to do this are:

- To group a set of steps as a single alternative beneath a BRANCH step. For details about this use of the SEQUENCE step, see "Using a SEQUENCE as the Target of a BRANCH" on page 223.
- To specify the conditions under which Integration Server will exit a sequence of steps without executing the entire set.

Using SEQUENCE to Specify an Exit Condition

In an implicit sequence, when a step fails, Integration Server automatically exits the sequence. (This is the same behavior as an explicit sequence for which the **Exit on** property is set to **FAILURE**.) By grouping steps into an explicit sequence, you can override this default behavior and specify the condition on which the SEQUENCE exits. To do this, you set the **Exit on** parameter as follows:

Set "Exit on" to…	If you want Integration Server to…
FAILURE	Exit the SEQUENCE when a step in the SEQUENCE fails. Execution continues with the next flow step in the flow service. This is the default behavior for a SEQUENCE.

Set "Exit on" to…	If you want Integration Server to…
	Exiting upon failure is useful if you have a series of steps that build upon one another. For example, if you have a set of steps that gets an authorization code and then submits a PO, you will want to skip the PO submission if the authorization step fails. When a SEQUENCE exits under this condition, the SEQUENCE step fails. **Note:** A failure by a transformer in a MAP step causes the containing SEQUENCE to exit when **Exit on** is set to **FAILURE**.
SUCCESS	Exit the sequence when any step in the SEQUENCE succeeds. Execution continues with the next step in the flow service. Exiting upon success is useful for building a set of alternative steps that are each attempted at run time. Once one of the members of the set runs successfully, the remaining steps in the SEQUENCE are skipped. If a child step in a SEQUENCE configured to exit on success fails, any changes that the child step made to the pipeline are rolled back (undone), and processing continues with the next child step in the SEQUENCE. **Note:** Successful execution by a transformer in a MAP step does *not* cause the containing SEQUENCE to exit when **Exit on** is set to **SUCCESS**. If all the child steps fail in a SEQUENCE configured to exit upon success, Integration Server considers the SEQUENCE step successful. **Note:** If you do not want a SEQUENCE configured to exit on success to be successful if all the child steps fail, insert an EXIT step as the last step in the SEQUENCE. Configure the EXIT step to exit from the flow and signal a failure. Specifically, set the **Exit from** property to $flow and the **Signal** property to FAILURE. At run time, if all the other child steps in the SEQUENCE fail and the child EXIT step succeeds, Integration Server exits the SEQUENCE with a failure condition. Integration Server throws an exception after the exiting the SEQUENCE.
DONE	Execute every step in the SEQUENCE. Integration Server considers a SEQUENCE step configured to exit on done to be successful as long as it executes all of its children within the specified time-out limit. The success or failure of a child step within the SEQUENCE is not taken into consideration. If a child step fails, any changes that it made to the pipeline are rolled back (undone), and processing continues with the next child step in the SEQUENCE.

Set "Exit on" to…	If you want Integration Server to…
	Integration Server considers a SEQUENCE step configured to exit on done to fail if all of the child steps do not execute within the specified time-out limit.

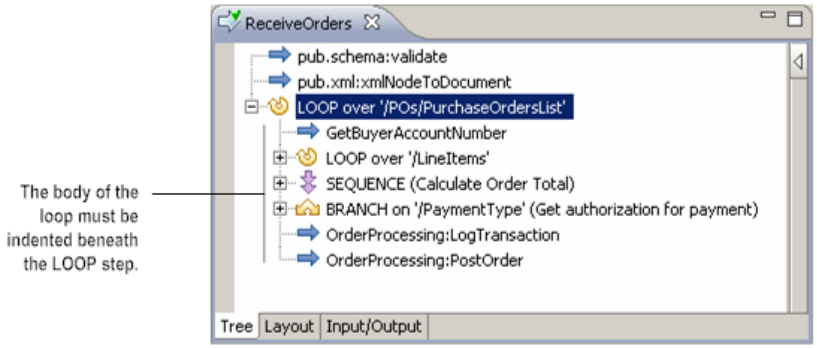
Note:
Rollback operations are performed on the first level of the pipeline only. That is, first-level variables are restored to their original values before the step failed, but the server does not roll back changes to any documents to which the first-level variables refer.

The LOOP Step

The LOOP step repeats a sequence of child steps once for each element in an array that you specify. For example, if your pipeline contains an array of purchase-order line items, you could use a LOOP step to process each line item in the array.

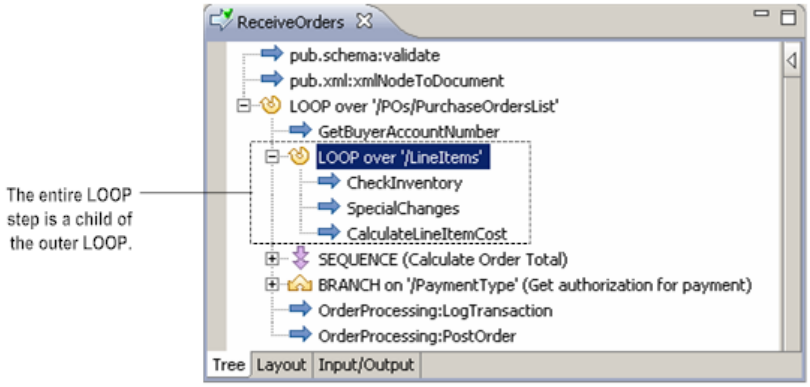
To specify the sequence of steps that make up the body of the loop (that is, the set of steps you want the LOOP to repeat), you indent those steps beneath the LOOP as shown in the following example.

Simple LOOP step



You may include any valid flow step within the body of a LOOP, including additional LOOP steps. The following example shows a pair of nested LOOPs. Note how the indentation of the steps determines the LOOP to which they belong.

Nested LOOP steps



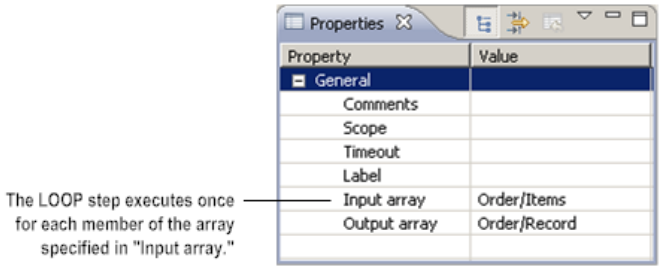
Specifying the Input Array

The LOOP step requires you to specify an input array that contains the individual elements that will be used as input to one or more steps in the LOOP. At run time, the LOOP step executes one pass of the loop for each member in the specified array. For example, if you want to execute a LOOP for each line item stored in a purchase order, you would use the document list in which the order's line items are stored as the LOOP's input array.

You specify the name of the input array on the LOOP step's Properties view. The array you specify can be any of the following data types:

- String list
- String table
- Document list
- Object list

LOOP properties



When you design your flow, remember that because the services within the loop operate against individual elements in the specified input array, they must be designed to take *elements* of the array as input, not the entire array.

For example, if your LOOP executes against a document list called *LineItems* that contains children called *Item*, *Qty*, and *UnitPrice*, you would specify *LineItems* as the **Input array** for the LOOP step, but services within the loop would take the individual elements of *LineItems* (for example, *Item*, *Qty*, *UnitPrice*, and so forth) as input.

Note:
The LOOP step is not thread-safe when the input array is a child of another variable (for example, a String list that is a child of a Document). Because the LOOP step changes the dimensionality of the input and output arrays during execution of the step, any threads invoking services that access the parent variable can experience the input array variable as either an array or a scalar. This results in unpredictable behavior for threads accessing the parent variable.

If the input array is a top-level variable in the pipeline, any thread that accesses the pipeline object (IData) for the service containing the LOOP step might also experience unpredictable behavior. Consequently, do not code other services that might concurrently access the object, such as a document, document list, or pipeline, that contains the input array. For information about the changes in dimensionality of inputs in a LOOP step, see "About the Pipeline for a LOOP Step" on page 236.

Collecting Output from a LOOP Step

If your LOOP step produces an output variable, the server can collect that output into an array in the pipeline.

To do this, you use the **Output array** parameter to specify the name of the array variable into which you want the server to collect output for each iteration of the loop. For example, if your loop checks inventory status of each line item in a purchase order and produces a String called *InventoryStatus* each time it executes, you would specify *InventoryStatus* as the value of **Output array**. At run time, the server will automatically transform *InventoryStatus* to an array variable that contains the output from each iteration of the loop.

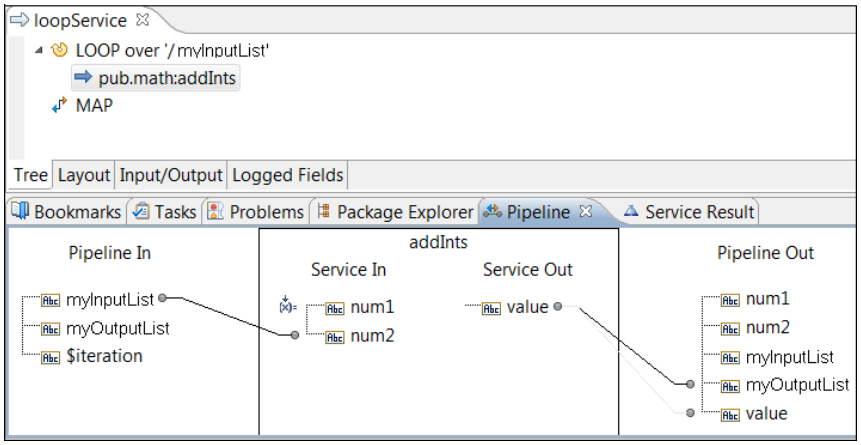
To collect output from each pass of the loop, specify the name of the output variable that you want the server to collect for each iteration.

About the Pipeline for a LOOP Step

In the body of a LOOP step, the field used as the input array is reduced dimensionally. For example, if the input array is a String list, the input is represented as a String within the body of the LOOP. If the input array is a String table, the input is a String list within the body of the LOOP. This is because a LOOP step operates on each element in the input array.

The field used as the output array is also reduced dimensionally within the body of a LOOP step. While the LOOP step produces an array, each iteration of the LOOP step produces one element in the array. If the output array is a String list, within the body of the LOOP it is a String. If the output array is a String table, within the body of the LOOP the output is a String list.

In the following example, the LOOP step executes the pub.math:addInts service for each item in the input array named *myInputList*. The LOOP step collects the output into an array named *myOutputList*. Inside the LOOP step, the pub.math:addInts service operates on one element of *myInputList* and produces one element of *myOutputList*. That is, the pub.math:addInts service takes a String as input and produces a String as output. Consequently, in the pub.math:addInts service pipeline, the input is a String named *myInputList* and the output is a String named *myOutputList*. If you viewed the pipeline after the LOOP step completes, *myInputList* and *myOutputList* would appear as String lists.



Building a LOOP Step

Use the following procedure to build a LOOP step in a flow service.

To build a LOOP step

1. If you are inserting a LOOP step into an existing flow service, display that service in the editor and select the step immediately above where you want the LOOP step inserted.

2. Do one of the following:

   - Click the ▾ button next to ⇨ on the flow service editor toolbar and click ↻.
   - Click ◁ by the side of the flow service editor to open the Palette view. Click ↻ and drag it to the flow service editor.

3. Complete the following fields on the Properties view:

For this property…	Specify…
Comments	An optional descriptive comment for this step.
Scope	The name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String.

For this property…	Specify…
	If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Label	An optional name for this specific LOOP step, or a null, unmatched, or empty string (**$null**, **$default**, blank). **Important:** If you use this step as a target for a BRANCH or EXIT step, you must specify a value in the **Label** property. For more information about the BRANCH and EXIT steps, see "The BRANCH Step" on page 218 or "The EXIT Step" on page 239.
Input array	The name of the array variable on which the LOOP will operate. This variable must be one of the following types: String list, String table, Document list, or Object list.
Output array	The name of the element that you want the server to collect each time the LOOP executes. You do not need to specify this property if the loop does not produce output values or if you are collecting the elements of **Input array**.

4. Build the body of the loop using the following steps:

   a. Insert a flow step using the buttons on the flow service editor toolbar.

   b. Indent the flow step using on the flow service editor toolbar to make it a child of the LOOP step.

   c. Set the properties for the child step as needed.

5. Use the Pipeline view to link the elements of the input array to the input variables required by each child of the LOOP step. For more information about using the Pipeline view, see "Mapping Data in Flow Services" on page 251.

Important:
When you build a LOOP step, make sure that you specify the output array variable in the LOOP **Output array** property *before* creating a link to the output array variable within a MAP or INVOKE step in the body of the LOOP. If you specify the output array variable *after* creating a link to it, the link will fail at run time. You can debug the step in Designer to see if the link succeeds. If the link fails, delete the link to the output array variable and then recreate it.

The EXIT Step

The EXIT flow step allows you to exit the entire flow service or a single flow step. You specify whether you want to exit from:

- The nearest ancestor (parent) LOOP or REPEAT flow step to the EXIT flow step.
- The parent flow step of the EXIT flow step.
- A specified ancestor flow step to the EXIT flow step.
- The entire flow service.

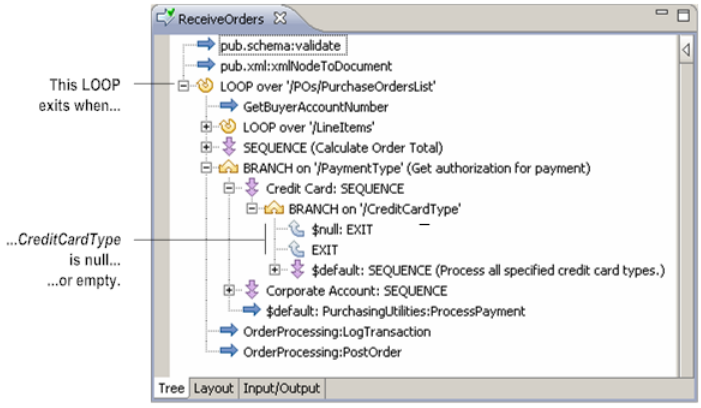
When you use the EXIT step, you indicate whether exiting should return a successful condition or a failure condition. If the exit is considered a failure, an exception is thrown. You can specify the text of the error message that is displayed by typing it directly or by assigning it to a variable in the pipeline.

Examples of when to use the EXIT step include to:

- Exit an entire flow service from within a series of deeply nested steps.
- Throw an exception when you exit a flow or a flow step without having to write a Java service to call Service.throwError( ).
- Exit a LOOP or REPEAT flow step without throwing an exception.

The following flow service contains two EXIT steps that, if executed, will exit the nearest ancestor LOOP step. If the value of *CreditCardType* is null or an empty string, the matching EXIT step executes and exits the LOOP over the '/PurchaseOrdersList' step.

Use the EXIT step to exit the nearest ancestor LOOP step



Building an EXIT Step

Use the following procedure to build an EXIT step in a flow service

To build an EXIT step

1. If you are inserting an EXIT step into an existing flow service, display that service in the editor and select the step immediately above where you want the EXIT step inserted.

2. Do one of the following:

   - Click the ▾ button next to ⇨ on the flow service editor toolbar and click ↳.
   - Click ◁ by the side of the flow service editor to open the Palette view. Click ↳ and drag it to the flow service editor.

3. Complete the following fields on the Properties view:

For this property…	Specify…
Comments	An optional descriptive comment for this step.
Label	An optional name for this specific step, or a null, unmatched, or empty string (**$null**, **$default**, blank). **Important:** If you use this step as a target for a BRANCH step, you *must* specify a value in the **Label** property. For more information about the BRANCH step, see "The BRANCH Step" on page 218.
Exit from	The flow step from which you want to exit. Specify one of the following: **Specify** — **To exit from the...** **$loop** — Nearest ancestor LOOP or REPEAT flow step. **$parent** — Parent flow step, regardless of the type of step. **$flow** — Entire flow. **Label** — Nearest ancestor flow step that has a label that matches this value. **Note:** If the label you specify does not match the label of an ancestor flow step, the flow will exit with an exception.
Signal	Whether the exit is to be considered a success or a failure. Specify one of the following: **Specify** — **To...** **SUCCESS** — Exit the flow service or flow step with a success condition. **FAILURE** — Exit the flow service or flow step with a failure condition. An exception is thrown after the exit. You specify the error message with the **Failure message** property.

For this property…	Specify…
Failure message	The text of the exception message you want to display. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols (for example, `%mymessage%`). The variable you specify must be a String. This property is not used when **Signal** is set to **SUCCESS**.

4. Click **File > Save**.

The MAP Step

The MAP step lets you adjust the contents of the pipeline at any point in a flow service. When you build a MAP step, you can:

- Prepare the pipeline for use by a subsequent step in the flow service by linking, adding, and dropping variables in the pipeline.
- Clean up the pipeline after a preceding step by removing fields that the step added but are not needed by subsequent steps.
- Move variables or assign values to variables in the pipeline.
- Initialize the input values for a flow service.
- Invoke several services (transformers) in a single step.
- Map documents from one format to another. For example, you can map a document in an XML format to an ebXML format or a proprietary format.

Tip:
The MAP step is especially useful for hard coding an initial set of input values in a flow service. To use it in this way, insert the MAP step at the beginning of your flow, and then use the **Set Value** modifier to assign values to the appropriate variables in **Pipeline Out**.

For more information about the MAP step, see "Mapping Data in Flow Services" on page 251.

11 Working in the Layout Tab

- What Is the Layout Tab 244
- What Does a Flow Service Look Like in the Layout Tab? 244
- Show or Hide the Grid Lines in the Flow Service Editor 247
- Building a Flow Service in the Layout Tab 247

The Layout tab is a graphical view of a flow service that Designer displays in the flow service editor. You use the Layout tab to create flow services.

What Is the Layout Tab

The Layout tab, like the Tree tab, is a view of a flow service that Designer displays in the flow service editor. You can use either page to build or edit a flow service. However, Layout tab provides a more graphical view in which to create flow services.

In the Layout tab, a flow service looks similar to a flow chart. Designer displays shapes for flow steps as well as for the start and end of the flow service. Lines connect the flow steps and show the order in which the flow steps execute.

Note: Designer uses the Tree tab as the default tab for building and viewing flow services. For this reason, unless specifically stated otherwise, the procedures in the *webMethods Service Development Help* are written for working in the Tree tab in the flow service editor.

When Should You Use Layout Tab?

Because the Layout tab and the Tree tab provide the same capabilities for building a flow service, work in whichever page you find easier to use. You can easily switch between tabs when building a flow service. For example, you might find it easier to insert the flow steps and define the basic structure of a flow service in the Layout tab, but use the Tree tab to perform data mapping.

You might prefer to use the Layout tab if:

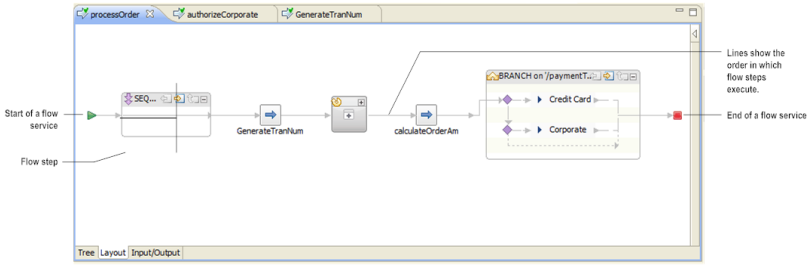
- You find that building a flow service as a flow chart is easier than building a flow service as a sequence of statements. You might be able to more easily envision the processes a flow service performs if you view the flow service as a diagram instead of as a series of line-by-line steps.
- You need to design a business process with someone unfamiliar with programming or unfamiliar with webMethods Integration Server. People who are not familiar with programming might be more comfortable with flow charts.
- You need to show diagrams of how the flow service works to management. (Flow services can be printed.)

What Does a Flow Service Look Like in the Layout Tab?

The Layout tab uses specific shapes and structures for the elements of a flow service, such as the start and end of a flow service, parent steps, child steps, and the order in which flow steps run. Designer displays flow steps sequentially from left to right, and executes steps in that order. (In tree view, Designer evaluates the target steps from top to bottom.)

The following illustration identifies the basic elements of a flow service in the Layout tab.

Basic elements of a flow service in the Layout tab



Designer automatically inserts the start and end symbols when you create a flow service. When you insert a step into a flow service, Designer automatically draws the lines connecting the flow step to the rest of the steps in the service.

Note: Designer automatically draws, redraws, and deletes lines when you insert, move, or delete steps in a flow service. You cannot move or delete lines.

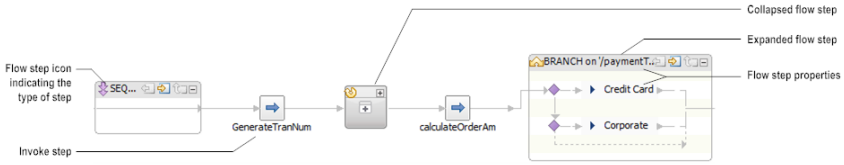
Viewing Flow Steps in the Layout Tab

In the Layout tab, the way in which flow steps are displayed depends on the type of step. For example, Designer displays an invoke step as a small square box in the flow service while more complex steps, such as BRANCH or LOOP, are displayed in larger boxes that you can collapse and expand as necessary to view and edit the step.

Tip:
When you move the mouse pointer over any flow step box in the Layout tab, the properties for the step appear in a tool tip.

Each box also displays an additional property that is relevant to the flow step type, such as Input array for LOOP and Switch for BRANCH.

Flow steps in the Layout tab



Each box that contains a flow step displays properties for the step, such as **Label** and **Comments**. The following table indicates which property is shown for each flow step.

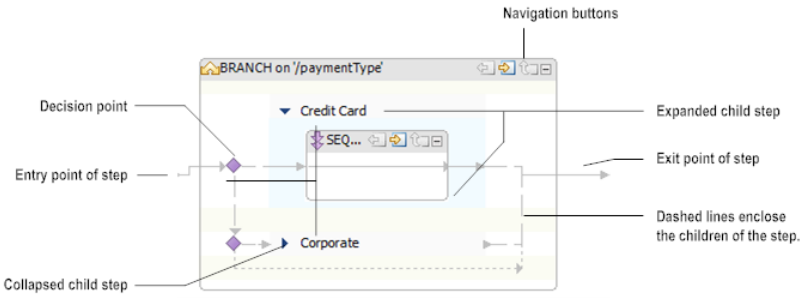
This step...	Displays this property...
BRANCH	**Switch** specifies the name of the variable whose value causes the execution of one of the BRANCH step's children at run time. If you branch on expressions, this property is blank.

This step...	Displays this property...
EXIT	**Signal** specifies whether the exit is considered a success or a failure. A SUCCESS condition exits the flow service or step. A FAILURE condition exits the flow service or step and throws an exception.
INVOKE	**Service** specifies the name of the service that is invoked at run time.
LOOP	**Input array** specifies the name of the array against which the selected LOOP step will run. Type the name of this variable exactly as it will appear in the pipeline at run time.
MAP	**Label**
REPEAT	**Repeat on**
SEQUENCE	none

Viewing Steps that Contain Child Steps in the Layout Tab

Steps such as BRANCH, LOOP, REPEAT, and SEQUENCE can contain child steps. Designer displays each step as a box in the flow service. Designer uses a solid line with arrows to indicate the path of data within the flow step. In a BRANCH step, Designer also uses dashed lines to enclose the children under the parent step. You can close these steps by clicking the minimize icon in the right corner.

Basic elements of a step that contains child steps in the Layout tab



The following table identifies the buttons and icons that you can use when building a flow step that contains child steps.

Button	Description
	Displays the step or child step in the editor while hiding the rest of the flow service. Use this button to view and edit the step or child step in isolation.
	Displays the previous view of the flow step or child step in the editor. Use this button to navigate back one level in the step or child step.
	Displays the full view of the flow service.

Button	Description
	Collapses the expanded step.
	Collapses the child step.
	Represents a point in the flow service where Designer evaluates the label or expression of the step and determines whether or not to execute the child step.

Show or Hide the Grid Lines in the Flow Service Editor

You might find that grid lines are helpful in the Layout tab when building or viewing a flow service. You can show and hide grid lines easily in Designer by selecting **View > Grid**. You can also use the Flow Service Editor preferences page to enable the grid and to customize grid line settings.

To enable and customize the grid in the flow service editor

1. In Designer: **Window > Preferences > Software AG > Service Development > Flow Service Editor**
2. In the Flow Services Editor preferences dialog box, under Grid Properties, select **Enable Grid**.
3. In the **Grid Width** and **Grid Height** text boxes, specify the size of the grid.
4. To save your changes, click **Apply.**
5. Click **OK**.

Building a Flow Service in the Layout Tab

Building a flow service in the Layout tab consists of the same stages as building a flow service in the Tree tab: creating the flow service, inserting flow steps, setting properties, declaring the service input and output parameters, mapping pipeline data, and setting run-time parameters. With the exception of how you insert and move flow steps, the procedures for completing each stage are the same in the Layout tab as they are in the Tree tab.

Tip:
You might find it easier to build services in the Layout tab if you have a larger view of the flow service. Use the and buttons on the Palette view to zoom in on or zoom out of the flow service.

Inserting a Flow Step

There are three ways in which you can insert flow steps into a flow service:

- Using the Palette view. You select the step in the Palette view and then drop it into the flow service editor. When you select a step and you move the mouse over the flow service in the editor, Designer displays a line to indicate where you can insert the step. To place a flow step, you click the area where you want to insert the step.
- Using the context menu. You right-click the area where you want to insert the step and then select the step in the context menu.
- Using the button on flow service toolbar. You click the button next to on the flow service editor toolbar and select the flow step that you want to insert.

Inserting a Flow Step Using the Palette View

To insert a flow step in the Layout tab using the Palette view

1. Open the service in which you want to insert a step. In the editor, make sure that you are in the Layout tab.
2. In the Palette view, select the step you want to insert.
3. In the editor, click the area where you want to insert the step.

   When you move the mouse pointer over the flow service, a line appears highlighting the places where you can insert the step. To insert a child step in a collapsed flow step, move the mouse pointer over the flow step box and click.
4. Designer inserts the step and automatically draws lines to connect the step to the rest of the steps in the flow service.
5. Click **File > Save**.

Inserting a Flow Step Using the Context Menu

To insert a flow step in the Layout tab using the context menu

1. Open the service in which you want to insert a step. In the editor, make sure that you are in the Layout tab.
2. In the editor, right-click the area where you want to insert the step. On the context menu, select **Insert** and then select the step you want to insert.

   If you right-clicked on an existing step, click **Insert** and then select **Before**, **After**, or **Into**, depending on where you want to place the step. Then, select the step you want to insert.
3. Designer inserts the step and automatically draws lines to connect the step to the rest of the steps in the flow service.

4. Click **File > Save**.

Notes for Inserting a Child Step into a BRANCH Step

When you build a BRANCH step in the Layout tab, keep the following points in mind:

- You can insert a flow step as a child (target) of the BRANCH, or insert a flow step into an existing target.
- In the Layout tab, Designer evaluates the target steps of a BRANCH from top to bottom. At run time, Designer executes the first target with a matching switch value or an expression that evaluates to true.
- A target with the **$default** label will be evaluated last, regardless of its position.
- A target without the **$default** label has a dashed line at the bottom of the BRANCH to indicate the flow of execution.

Changing the Order of Steps in a Flow Service

In the Layout tab, the arrows connecting the flow steps indicate the order in which the steps execute. You can move, or relocate, steps in a flow service to change the order in which steps execute. You can also relocate a step to make it a child of another step in the flow service.

To change the order of a step in a flow service on the Layout tab

1. On the Layout tab, select the step whose execution order you want to change.
2. Use the **Cut**, **Copy**, and **Paste** buttons on the toolbar or the **Cut**, **Copy**, and **Paste** options on the context menu to change the location of the flow step.

   For more information, see "Moving and Copying Elements" on page 48.

12 Mapping Data in Flow Services

- What Does the Pipeline View Contain? 252
- Basic Mapping Tasks 256
- About Linking Variables 256
- About Assigning Values to Pipeline Variables 271
- Dropping Variables from the Pipeline 275
- Adding Variables to the Pipeline 276
- Working with Transformers 278
- Testing Flow Steps Before Running a Flow Service 285
- Mapping Using ForEach 286

Because systems rarely produce data in the exact format that other systems need, you commonly need to build flow services that perform data transformations. Data transformation resolves differences in the way data is represented within documents that applications and systems exchange. In Designer, data transformations can be accomplished by mapping data. By mapping, you can accomplish the following types of transformations:

- **Name transformations.** This type of transformation resolves differences in the way data is named. For example, one service or document format might use *telephone* as the name of the variable for telephone number information and another might use *phoneNumber*. When you perform name transformations, the value and position of a variable in the document structure remains the same, but the name of the variable changes.
- **Structural transformations.**This type of transformation resolves differences in the data type or structure used to represent a data item. For example, one service or document format might put the telephone number in a String called *telephone*, and the next may expect to find it nested in a Document named *customerInfo*. When you perform structural transformations, the value of the variable remains the same, but the data type or position of the variable in the Document structure changes.
- **Value transformations.**This type of transformation resolves differences in the way values are expressed (for example, when systems use different notations for values such as standard codes, units of currency, dates, or weights and measures). When you perform value transformations, the name and position of the variable remain the same, but the data contained in the variable changes. For example, you can change the format of a date, concatenate two Strings, or add the values of two variables together.

When you build flow services or convert between document formats, you may need to perform one, two, or all of the above types of data transformation. The webMethods flow language provides two ways for you to accomplish data transformations between services and document formats in the pipeline: you can map variables to each other (create links) or you can insert transformers, which are services invoked within a MAP step.

What Does the Pipeline View Contain?

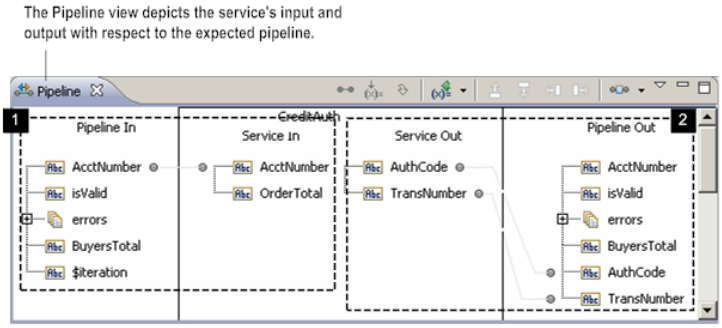
The Pipeline view offers a graphical representation of all of your data through which you can map data and inspect the contents of the pipeline. You use the tools on this view to route variables (data) between services or between document formats.

The Pipeline view displays the pipeline for invoked services (INVOKE steps) or MAP steps in a flow service. The contents of the Pipeline view are different for INVOKE steps than for MAP steps.

Pipeline View for an INVOKE Step

For an INVOKE step, the Pipeline view depicts two stages of the pipeline with respect to the selected service in the editor.

Pipeline view for an INVOKE step



This stage...	Represents...
1	The expected state of the pipeline just before the selected service executes. **Pipeline In** depicts the set of variables that are expected to be in the pipeline before the service executes (based on the declared input and output parameters of the preceding services). **Service In** depicts the set of variables the selected service expects as input (as defined by its input parameters). In the Pipeline view, you can insert "pipeline modifiers" at this stage to adjust the contents of the pipeline to suit the requirements of the service. For example, you can link variables, assign values to variables, drop variables from the pipeline, or add variables to the pipeline. Modifications that you specify during this stage are performed immediately *before* the service executes at run time.
2	The expected state of the pipeline just after the service executes. **Service Out** depicts the set of variables that the selected service produces as output (as defined by its output parameters). **Pipeline Out** depicts the set of variables that are expected to be in the pipeline after the service executes. It represents the set of variables that will be available to the next service in the flow. If the selected service (INVOKE step) is the last step in the flow service, **Pipeline Out** displays the output variables for the flow service (as declared on the Input/Output tab). In the Pipeline view, you can insert "pipeline modifiers" at this stage to adjust the contents of the pipeline. For example, you can link variables, assign values to variables, drop variables from the pipeline, or add variables to the pipeline. Modifications that you specify during this stage are performed immediately *after* the service executes at run time.

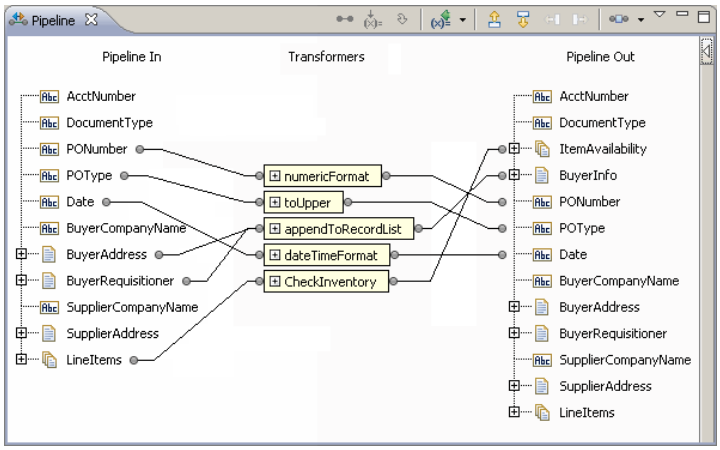
Note: Designer displays small symbols next to a variable icon to indicate validation constraints. Designer uses ▫ to indicate an optional variable. Designer uses the ‡ symbol to denote a variable with a content constraint. Designer also uses ▲ to indicate that the variable has a default value that can be overridden assigned to it and 🔒 to indicate that the variable has a null value that cannot be overridden assigned to it. A combination of the ▲ and 🔒 symbols next to a variable icon indicates that the variable has a fixed default value that is not null and cannot be overridden.

For information about applying constraints to variables, see "About Variable Constraints" on page 607.

Pipeline View for a MAP Step

For a MAP step, the Pipeline view displays a single stage of the pipeline. The Pipeline view contains two sets of variables: **Pipeline In** and **Pipeline Out**. Between these sets of variables, the Pipeline view contains a column named **Transformers**.

Pipeline view for a MAP step



- The **Pipeline In** column represents input to the MAP step. It contains the names of all of the variables in the pipeline at this point in the flow.
- The **Transformers** column displays any services inserted in the MAP step to complete value transformations. For more information about invoking services in a MAP step, see "Working with Transformers" on page 278.
- The **Pipeline Out** column represents the output of the MAP step. It contains the names of variables that will be available in the pipeline when the MAP step completes.

When you first insert a MAP step into your flow, **Pipeline In** and **Pipeline Out** are identical. However, if the MAP step is the only step in the flow service or is the last step in the flow service, **Pipeline Out** also displays the variables declared as output in the flow service.

Scrolling in Pipeline View

By default, you scroll vertically or horizontally through the entire Pipeline view. However, you can enable horizontal and vertical scrolling for each column in Pipeline view. Independent scrolling is especially useful when mapping a large amount of data in the Pipeline view.

To enable independent scrolling in Pipeline view

- Right-click anywhere inside the Pipeline view and select **Enable independent scrolling** to scroll through each column in Pipeline view horizontally and vertically independent of other columns.

  Tip:
  While scrolling through a large amount of data, if you do not want Designer to display the links if the source or target variables are not visible, right-click anywhere inside the Pipeline view and select **Hide links if variables are not visible**.

Viewing Full Namespace Path of Referenced Document Types

By default, Designer displays only the names of the document reference or document reference list variables in Pipeline view. However, you can choose to view or hide the full namespace path of the referenced document types in Pipeline view. If you choose to view the full namespace path of the referenced document types in Pipeline view, they appear in parenthesis after the name of the document reference or document reference list variables, for example, contactInfo (DocTypesAndSpecs:address)

The **Show Referenced Document Type Name** setting applies to:

- All document reference or document reference list variables in Pipeline view and not just the selected nodes.
- All MAP and INVOKE steps of all flow services.

This setting remains in effect even after you shutdown and restart Integration Server.

Note:
You can also use the Flow Service Editor preferences page to view or hide the full namespace path of the referenced document types in Pipeline view.

To view or hide the full namespace path of the referenced document types in Pipeline view

- Right-click anywhere inside the Pipeline view and select **Show Referenced Document Type Name**.

Printing the Pipeline

The following procedure describes how to use the **View as HTML** command to produce a printable version of the pipeline.

To print the pipeline

1. In the Package Navigator view, open the flow service for which you want to print the pipeline.
2. In the editor, select the INVOKE or MAP step for which you want to print the pipeline.

3. Scroll or resize the Pipeline view to display the portion of the pipeline you want to view as HTML.

4. Right-click anywhere inside the Pipeline view and click **View as HTML**.

   Designer creates an HTML page and displays it in your default browser.

5. Use your browser's print command to print the pipeline.

Basic Mapping Tasks

Basic mapping tasks are the tasks you perform to manage the pipeline contents and the values of variables in the pipeline. In the Pipeline view, you can perform the following basic mapping tasks:

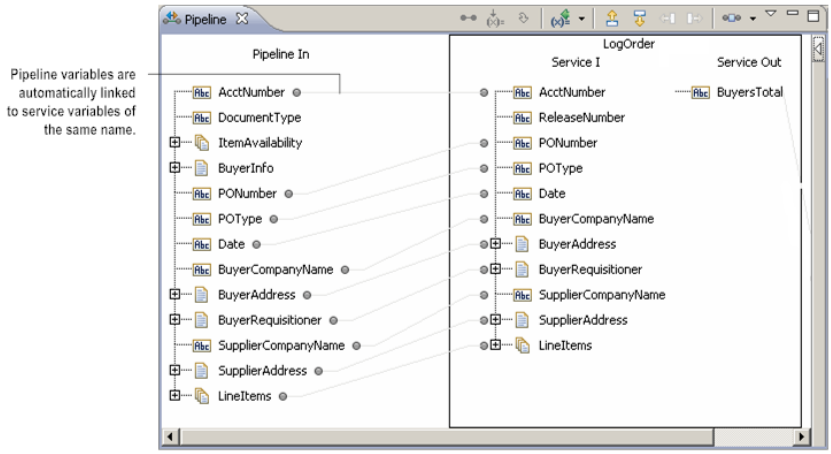
- **Link variables to each other.** You can copy the value of a variable in one service or document format to a variable in another service or document format.
- **Assign values to variables.** You can hard code variable values or assign a default value to variables.
- **Drop variables from the pipeline.** You can remove pipeline variables that are not used by subsequent services in a flow.
- **Add variables to the pipeline.** You can add variables that were not declared as input or output parameters of the flow service. You can also add input and output variables for services that the flow service invokes (internally invoked services).

About Linking Variables

When you want to copy the value of a variable in a service or document format to another variable, you link the variables. Designer connects service and pipeline variables in the Pipeline view with a line called a *link*. Creating a link between variables copies the value from one variable to another at run time.

Within a flow, Designer implicitly links variables whose names are the same and whose data types are compatible. For example, the service in the following flow takes a variable called *AcctNumber*. Because a variable by this name already exists in **Pipeline In**, it is automatically linked to the *AcctNumber* variable in **Service In**. Designer connects implicitly linked variables with a gray link.

Implicit links between pipeline and service variables



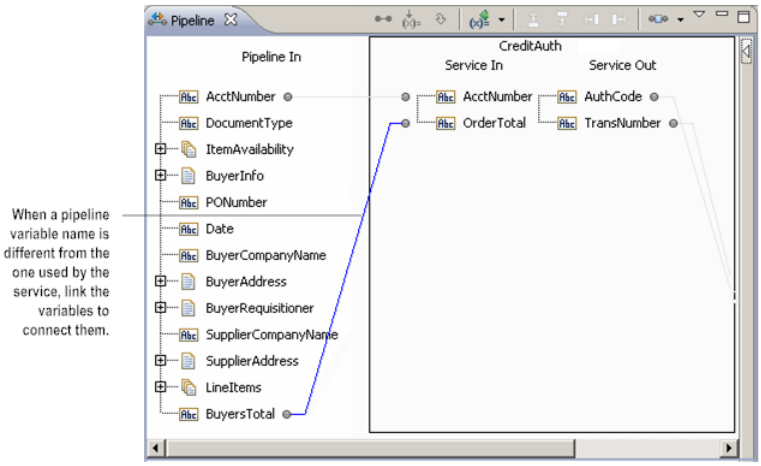
Note:
The Pipeline view does not display implicit links for a MAP step.

In cases where the services in a flow do not use the same names for a piece of information, use the Pipeline view to ***explicitly link*** the variables to each other. Explicit linking is how you accomplish name and structure transformations required in a flow. Designer connects explicitly linked variables with a solid black line.

On the input side of the Pipeline view, use ⊶ to link a variable from the pipeline to the service. In the following example, the service expects a value called *OrderTotal*, which is equivalent to the pipeline variable *BuyersTotal* (that is, they are simply different names for the same data). To use the value of *BuyersTotal* as the value for *OrderTotal*, you "link" the pipeline variable to the service using ⊶.

At run time, the server will copy the value from the source variable (*BuyersTotal*) to the target variable (*OrderTotal*) before executing the service.

Linking the pipeline to service input



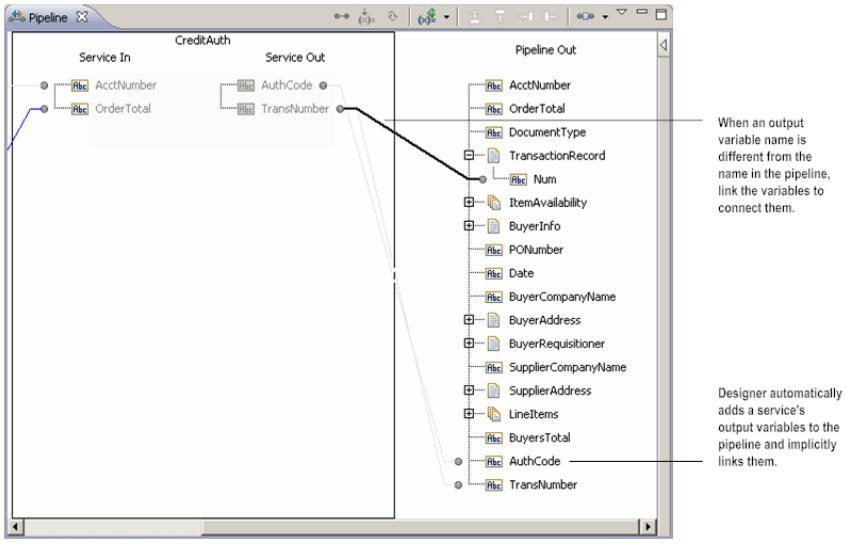
Important:

Do not link variables with different Object constraints. If you link variables with different Object constraints and input/output validation is selected, the run-time result is undefined.

All the output variables that a service produces are automatically placed in the pipeline. Just as you can link variables from the **Pipeline In** stage to a service's input variables, you can link the output from a service to a different variable in **Pipeline Out**.

In the following example, a variable called TransNumber is linked to the field Num in a Document called *TransactionRecord*. At run time, the server will copy the value of *TransNumber* to *Num*, and both *TransNumber* and *Num* will be available to subsequent services in the flow.

Linking service output to the pipeline



Creating a Link Between Variables

When you link variables in the pipeline, keep the following points in mind:

- The variable that you are linking from is the *source*. For example, when you link a variable in **Pipeline In** to one in **Service In**, the **Pipeline In** variable is the source. When you link a variable in **Service Out** to one in **Pipeline Out**, the **Service Out** variable is the source.
- The variable you are linking to is the *target*. For example, when you link a variable in **Pipeline In** to one in **Service In**, the **Service In** variable is the target. When you link a variable in **Service Out** to one in **Pipeline Out**, the **Pipeline Out** variable is the target.
- A **Service In** variable can be the target of more than one link only if you use array indexing or if you place conditions on the links to the variable.
- By linking variables to each other, you are copying data *from* the source variable to the target variable. (Documents, however, are copied by reference. For more information, see "What Happens When Integration Server Executes a Link?" on page 260.)
- Target variables can be connected to only one source variable. After you draw a link to a target variable, you cannot draw another link to the target variable. (Two exceptions to this rule

involve array variables and conditional links. For more information about linking array variables, see "Linking to and from Array Variables in the Pipeline" on page 264. For more information about placing conditions on links between variables, see "Linking Variables Conditionally" on page 270.

- You cannot create a link to a variable if you already assigned a value to a variable.
- After a link executes, both the source and target variables exist in the pipeline. The target variable does not replace the source variable.
- You cannot create a link to a variable if the variable has a fixed null or default value assigned to it. Designer uses the symbol next to the variable icon to indicate that the variable has a fixed value that you cannot override by linking it to another variable.

To create a link between variables

1. In the flow service editor, select the INVOKE or MAP step containing the variables you want to link.
2. Open the Pipeline view.
3. If you want to create a link between a variable in **Pipeline In** and one in **Service In**, do the following:
   a. In **Pipeline In**, click the pipeline variable you want to use as the source variable.
   b. In **Service In**, click the input variable you want to use as the target variable.
   c. Click on the Pipeline view toolbar.
4. If you want to create a link between a variable in **Service Out** and one in **Pipeline Out**, do the following:
   a. In **Service Out**, click the output variable you want to use as the source variable.
   b. In **Pipeline Out**, click the pipeline variable you want to use as the target variable.
   c. Click on the toolbar.
5. Click **File > Save**.

Notes:

- If the variable types are incompatible and cannot be linked to one another, Designer prevents you from creating a link between the variables and displays a message stating that the operation is not allowed.

- If you created a link to or from an array variable, you must specify which element in the array you are linking to or from. For more information about array linking, see "Linking to and from Array Variables in the Pipeline" on page 264.
- If you want to place a condition on the execution of the link, see "Linking Variables Conditionally" on page 270.
- Do not link variables with different Object constraints. If you link variables with different Object constraints and input/output validation is selected, the run-time result is undefined.

Tip:
You can also use your mouse to link variables to one another. To do this, select the source variable and drag your mouse to the appropriate target variable.

What Happens When Integration Server Executes a Link?

When executing a link between variables at run time, Integration Server does one of the following:

- Copies the value from the source variable to the target variable. For example, when you link a source String variable to a target String variable, Integration Server copies the value of the source String to the target String. This is called "copying by value."
- Creates a *reference* to the source variable and uses the reference as the value of the target variable. For example, when executing a link between a source Document and a target Document, Integration Server creates a reference to the source Document value and uses the reference as the value of the target Document. This is called "copying by reference."

Integration Server copies by value when the source or target variable is a String. (An exception to this behavior is that when executing a link from a String to an Object, the Integration Server copies by reference.)

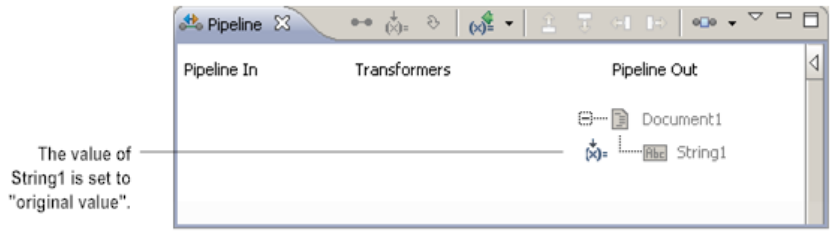
When executing links between all other types of variables, Integration Server copies by reference. Copying by reference significantly reduces the memory and time required for executing a link at run time.

When a value is copied by reference, any changes you make to the value of the source variable in subsequent flow steps affect the target variable. This is because the value of the source variable is the value of the target variable. The target variable does not contain a copy of the source variable value. If, in a later flow step, you used to assign a value to the source variable, you would be changing the value of the target variable as well. (The target variable references the value of the source variable.)

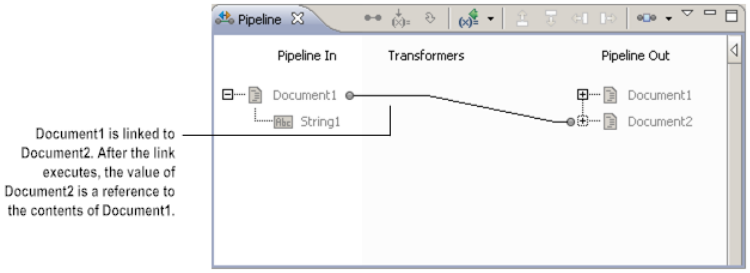
Example of Copying By Reference

The following images show a series of MAP steps in a flow service. In this example, the value of the source variable is changed after the link to the target variable executes. This action changes the value of the target variable as well.

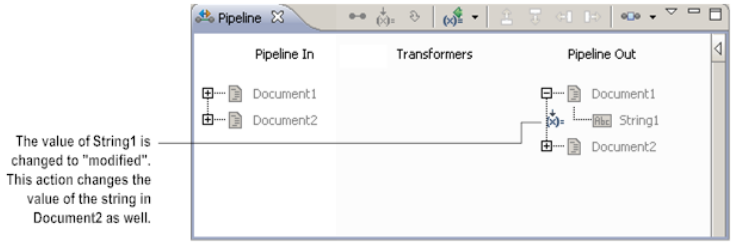
Step 1: The value of String1 is set to "original value"



Step 2: Document1 is linked to Document2

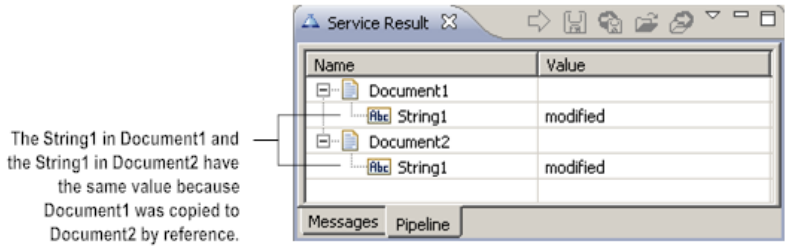


Step 3: The value of String1 is changed to "modified" after the link executes



When this flow service executes, it returns the following results.

Results of flow service



In Step 3, the value of the String1 in Document1 was set to "modified." However, the value of String1 in Document2 changed also. This is because in Step 2 of the flow service, the value of Document1 was copied to Document2 by reference. Changes to the value of Document1 in later flow steps also change the value of Document2.

Preventing Pipeline Values from Being Overwritten

To prevent the value of the target variable from being overwritten by changes to the value of the source value in subsequent steps in the flow service as demonstrated in "Example of Copying By Reference" on page 260, you can do one of the following:

- When working with Document variables, link each child of the Document variable individually. This method can be time consuming and might significantly increase the memory and time required to run the service. However, this might be the best approach if the target Document variable needs only a few values from the source Document variable.
- After you link the source variable to a target variable, use the **Drop** modifier to drop the source variable. Only the target variable will have the reference to the data. This method ensures that the value of the target variable will not be overwritten in a subsequent step, but does not increase the memory and time required to execute the service.
- Create a service that performs a copy by value. Insert this service (as an INVOKE step or as a transformer) and link the variables to the service instead of linking them to each other. (In the case of Document variables, you could create a Java service that clones the IData object underlying the Document.) In situations where you link one Document variable to another, using a "cloning" service would require less time than linking the contents of a Document variable field by field.

Linking to Document and Document List Variables

When working with Document variables in the pipeline, you can link a source variable to the Document variable or to the children of the Document variable. Keep the following points in mind when linking to or from Document, Document List, Document Reference, or Document Reference List variables:

- A Document (or a Document List) and its children cannot both be targets. After a Document or Document List is the target of a link, its children cannot be the targets of links.
- After the child variable of a Document or Document List is the target of a link, the parent Document or Document List cannot be a target of a link.
- If you link from a Document variable to another Document variable, the structure of the source Document variable overwrites the structure of the target Document variable.
- You cannot link a nested Document List to a target Document List when the Document Lists have different sizes. A nested Document List is one that is contained within a parent Document List. Document Lists are considered to have different sizes when they have a different number of entries within the lists. If you need to move values from the source Document List to the target, create user code that uses a LOOP flow step to assign values from the source to the target one by one.
- When a Document Reference or Document Reference List refers to an IS document type that contains identically named variables that are of the same data type and both identically named variables are assigned a value or are linked to another variable, Integration Server might not maintain the order of the document contents in the pipeline when the service executes. For example, Integration Server might group all of the identical variables at the end of the document. To prevent the change in the order of document contents, set default values for the identically named variables. To do this, insert a MAP step in the service before the step in which you want

link or assign a value to the variables. In the MAP step, under Pipeline Out, select the Document Reference variable and click on the Pipeline view toolbar. In the Enter Input for dialog box, assign default values to the identically named variables.

Linking Variables of Different Data Types

In the Pipeline view, you can link different, but compatible, data types to one another. For example, you could link a String value called *AccountNumber* to a String List called *Accounts*. At run time, the server automatically performs the structural transformation necessary to link the data in *AccountNumber* to *Accounts*. (In this case, the transformation will result in a single-element String List.) By linking different data types to one another, you can perform structural transformations.

If you link variables of different data types, keep the following points in mind:

- Not all data types can be linked to one another. You cannot link a Document (IData object) to a String, for instance. If two data types are incompatible, Designer will not allow you to link them to each other.
- You can only link a variable to another variable of the same ***primitive type***. The primitive type refers to the data type of the variable when all dimensionality is removed. For example, the primitive type for a String List or a String Table would be String. Two exceptions to this rule are the following:
  - Any variable can be linked to an Object or an Object List variable
  - An Object can be linked to any data type.

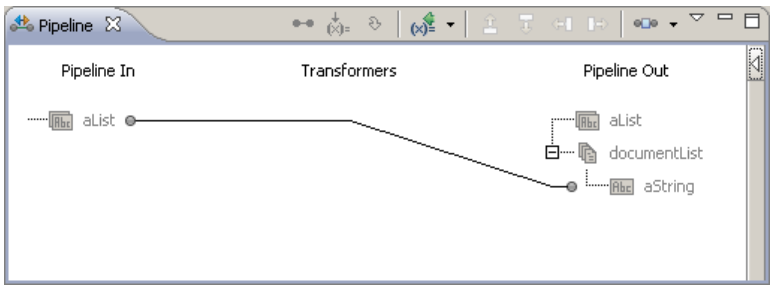
  If there is a type mismatch between the Object or Object List and the other variable at run time, Integration Server does not execute the link.
- Object and Object List variables constrained with an assigned Java class should be linked only to other Object and Object List variables of the same Java class or to Object and Object List variables of unknown type. Although Designer permits a link between constrained Objects with different Java classes, the run-time result is undefined. For more information about specifying Java classes for Objects, see "Java Classes for Objects" on page 1085.
- When you link between scalar and array variables, you can specify which element of the array variable you want to link to or from. ***Scalar variables*** are those that hold a single value, such as String, Document, and Object. ***Array variables*** are those that hold multiple values, such as String List, String Table, Document List, and Object List. For example, you can link a String to the second element of a String List. Alternatively, you can link the second element in a String List to a String.
- When you link between scalar and array variables and you do not specify which element in the array variable that you want to link to or from, Designer uses the default behavior to determine the value of the target variable.

Converting a String List to a Document List in the Pipeline

You can convert a String List to a Document List in the pipeline by mapping a String List to a String in a Document List. In the following image, *aList* is the String List you want to convert to

a Document List. The variable *documentList* is the Document List to which you want to copy the values contained in the String List. *documentList* has a String child *aString*. To convert the String List to a Document List, link *aList* to *aString*.

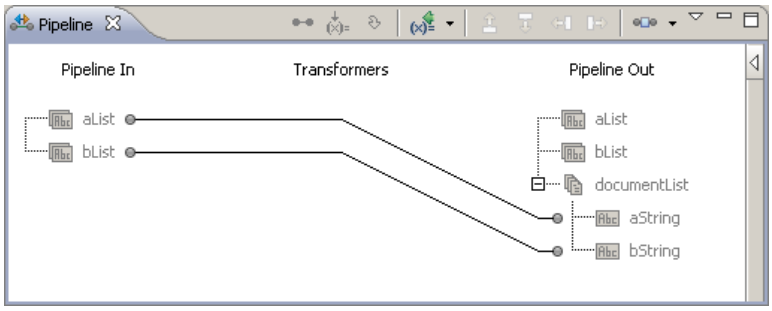
Converting a String List to a Document List



Converting Two String Lists to a Document List in the Pipeline

Two String Lists can be combined into one Document List in the pipeline by linking each String List to a String nested in a Document List. For example, suppose that you had String List variables named *aList* and *bList*, and *documentList* had two String children named *aString* and *bString*. You could combine the two String Lists by linking *aList* to *aString* and *bList* to *bString*.

Converting two String Lists to a Document List



Tip:
You can also convert a String List to a Document List (IData[ ] object) by invoking the built-in service pub.list:stringListToDocumentList. You can insert the service as an INVOKE step or as a transformer. For more information about transformers, see "Working with Transformers" on page 278. For more information about built-in services, see the *webMethods Integration Server Built-In Services Reference*.

Linking to and from Array Variables in the Pipeline

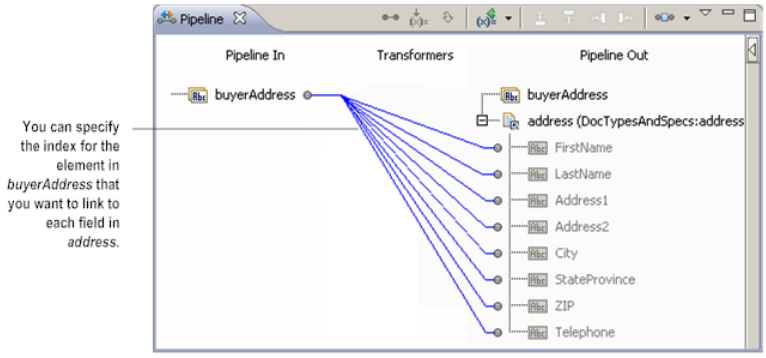
When you link to or from an array variable (String List, String Table, Document List, or Object List), you can specify which element in the array you want to link to or from. After you link the variables, you specify the index that represents the position of the element in the array.

- For String Lists and Object Lists, you can specify the index for the list element you want to link. For example, you can link the third element in a String List to a String.

- For String Tables, you can specify the row and column indexes for the cells you want to link. For example, you can link the value of the element in the third column of the second row of a String Table to a String.
- For Document Lists, you can specify the index for the Document that you want to link. For example, you can link the second Document in a Document List to a Document variable.
- For a variable in a Document List, you can specify the index of the Document that contains the value that you want to link to or from. For example, if the Document List *POItems* contains the String *ItemNumber*, you can link the *ItemNumber* value from the second *POItems* Document to a String variable.

For example, suppose that a buyer's address information is initially stored in a String List. However, the information might be easier to work with if it were stored in a Document. To map the information in the String List to a Document, create a link between the String List and each field in the Document. Then, specify an index value for each link. In the following pipeline, the elements in *buyerAddress* String List are mapped to the *address* Document.

You can specify an index value when linking to or from an array variable



Note: Designer uses blue links in the Pipeline view to indicate that properties (conditions or index values for arrays) have been applied to the link between variables.

Creating a Link to or from an Array Variable

When you are linking to or from an array variable, keep the following points in mind:

- To link to or from an element in an array variable, you need to know the index for the element's position in the array. Array index numbering begins at 0 (the first element in the array has an index of 0, the second element has an index of 1, and so on).
- To dynamically specify the index, you can set the index to the value of a pipeline variable. The variable you specify must be a String. To use a pipeline variable, specify the variable name enclosed in percent signs (%). For example, if the pipeline contains the variable *itemNumber* that will contain the index you want to use at run time, specify `%itemNumber%` for the index. For the link to execute successfully at run time, the value of the variable must be a non-negative integer.
- If you link to an array variable and specify an index that does not exist, Designer increases the length of the array to include the specified array index. For example, suppose that a String

List has length 3. You can link to the String List and specify an index of 4; that is, you can link to the fifth position in the String List. At run time, the Integration Server increases the length of the String List from 3 to 5.

- Each element in an array can be the source or target of a link; that is, each element in the array can be the start or end of a link. For example, if a source String List variable contains three elements, you can link each of the three elements to a target variable.
- If the source *and* target variables are arrays, you can specify an index for each variable. For example, you can link the third element in a source String List to the fifth element in target String List.
- If you do not specify an array index for an element when linking to or from arrays, the default behavior of the Pipeline view will be used. For information about the default behavior of the Pipeline view, see "Default Pipeline Rules for Linking to and from Array Variables" on page 267.
- If you are linking to or from a String Table, you need to specify an index value for the row and column.
- When you link a Document or Document List variable to another Document or Document List variable, the structure of the source variable determines the structure of the target variable. For more information, see "Linking to Document and Document List Variables" on page 262.
- At run time, the link (copy) fails if the source array index contains a null value or if you specify an invalid source or target index (such as a letter or non-numeric character). Integration Server generates journal log messages (at debug level 6 or higher) when links to or from array variables fail.

The following procedure explains how to link to or from an array variable.

To create a link to or from an array variable

1. Create a link between the variables using the procedure described in "Creating a Link Between Variables" on page 258.
2. In Pipeline view, click the link that connects the variables.
3. In the Properties view, click the **Indices** value and click ... . The Link Indices dialog box appears.
4. If the source variable is an array variable, under **Source**, type the index that contains the value you want to link. If the source variable is a String Table, you need to specify a row index and a column index.
5. If the target variable is an array variable, under **Destination**, type the index to which you want to link the source value. If the target variable is a String Table, you need to specify a row index and a column index.
6. Click **OK**.

Tip:

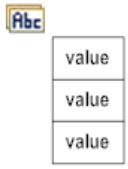
You can also open the Link Indices dialog box by selecting the link between the variables and clicking on the Pipeline view toolbar.

Default Pipeline Rules for Linking to and from Array Variables

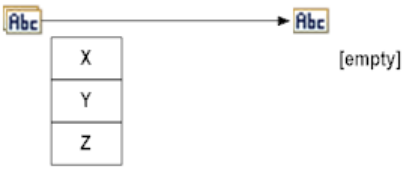
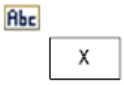
When you create links between scalar and array variables, you can specify which element of the array variable you want to link to or from. *Scalar variables* are those that hold a single value, such as String, Document, and Object. *Array variables* are those that hold multiple values, such as String List, String Table, Document List, and Object List. For example, you can link a String to the second element of a String List.

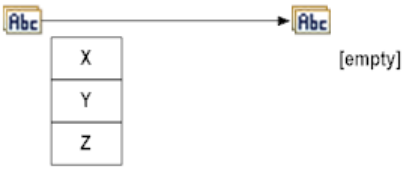
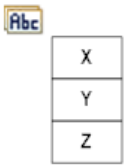
If you do not specify which element in the array variable that you want to link to or from, Designer uses the default rules in the Pipeline view to determine the value of the target variable. The following table identifies the default pipeline rules for linking to and from array variables.

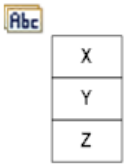
If you link…	To…	Then…
A scalar variable	An array variable that is empty (the variable does not have a defined length)	The link defines the length of the array variable; that is, it contains one element and has length of one. The first (and only) element in the array is assigned the value of the scalar variable.

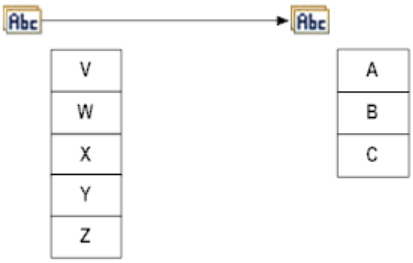
If you link…	To…	Then…
A scalar variable	An array variable with a defined length	The length of the array is preserved and each element of the array is assigned the value of the scalar variable.
		

If you link…	To…	Then…
An array variable	A scalar variable	The scalar variable is assigned the first element in the array.

If you link…	To…	Then…
		

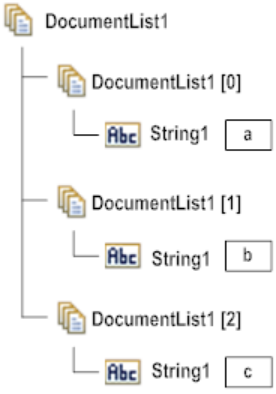
If you link…	To…	Then…
An array variable	An array variable that does not have a defined length	The link defines the length of the target array variable; that is, it will be the same length as the source array variable. The elements in the target array variable are assigned the values of the corresponding elements in the source array variable.
		

If you link…	To…	Then…
An array variable	An array variable that has a defined length	The length of the source array variable *must* equal the length of the target array variable. If the lengths do not match, the link will not occur. If the lengths are equal, the elements in the target array variable are assigned the values of the corresponding elements in the source array variable.
		

If you link…	To…	Then…
		No link occurs.

A source variable that is the child of a Document List is treated like an array because there is one value of the source variable for each Document in the Document List. For example:

If you link…	To…

Where the value of DocumentList1 is...	Then the value of StringList1 is…
	

Deleting a Link Between Variables

When you delete link in Pipeline view, the variables are no longer linked. Designer also deletes any properties you applied to the link.

To delete a link between variables

1. In the flow service editor, select the INVOKE or MAP step containing the variables with the link you want to delete.

2. In the Pipeline view, select the link that you want to delete.

3. Click **Edit > Delete**.

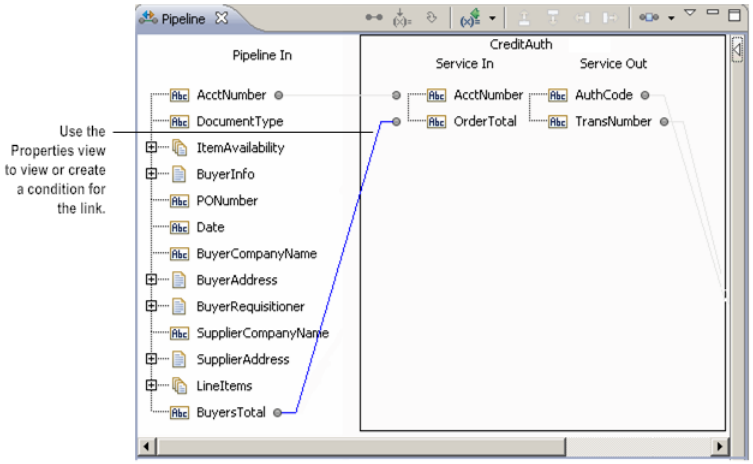
Tip:
You can also delete a link by selecting it and then pressing the DELETE key.

Linking Variables Conditionally

You can place conditions on the links you draw between variables. At run time, Integration Server evaluates the condition and executes the link (copies the value) only if the condition evaluates to true.

A condition consists of one or more expressions that you write using the syntax that Designer provides. An expression can check for the existence of a variable in the pipeline, check for the value of a variable, or compare a variable to another variable. For example, in the following service, you might want to link the *BuyersTotal* variable in **Pipeline In** to the *OrderTotal* variable in **Service In** only if the *BuyersTotal* has a value that is not null. After you link two variables, you would edit the properties and add the condition that needs to be true.

A blue link indicates that a condition is applied to the link connecting the variables



Designer uses a blue link in the Pipeline view to indicate that properties (that is, conditions or index values for arrays) have been applied to a link between variables.

Note:
You cannot add conditions to the links between implicitly linked variables.

Linking Multiple Source Variables to a Target Variable

By applying conditions to the links between variables, you can link more than one source variable to the same target variable. When you draw more than one link to the same target variable, at most, only *one* of the conditions you apply to the links can be true at run time. The conditions must be mutually exclusive.

At run time, Integration Server executes all conditional links whose conditions evaluate to true. If more than one conditional link to the same target variable evaluates to true, the value of the target variable will be the result of whichever link executes last. Because the order in which links are executed at run time is not guaranteed, the final value of the target variable may vary.

Tip:
If the conditions for links to the same target variable are not mutually exclusive, consider using a flow service containing a BRANCH step instead. In BRANCH steps, child steps are evaluated in a top to bottom sequence. Integration Server executes the first child step that evaluates to true and skips the remaining child steps. For more information about the BRANCH step, see "The BRANCH Step" on page 218.

Applying a Condition to a Link

Keep the following points in mind when making a conditional link:

- You can only add conditions to the links between explicitly linked variables. You cannot add conditions to the links between implicitly linked variables.
- When drawing more than one link to the same target variable, make sure that the conditions assigned to each link are mutually exclusive.
- You can temporarily disable the condition placed on a link. For more information, see "Disabling and Enabling Conditions" on page 443.

To apply a condition to the link between variables

1. Create a link between the variables using the procedure described in "Creating a Link Between Variables" on page 258.
2. In Pipeline view, click the link that connects the variables.
3. In the Properties view, set the **Evaluate copy condition** property to **True**.
4. In the **Copy condition** property text box, type the condition you want to place on the link. For information about the syntax used in conditions, see "Conditional Expressions" on page 1113.

About Assigning Values to Pipeline Variables

You can assign values to variables in **Service In** or **Pipeline Out** using on the Pipeline view toolbar. When you assign a value to a variable, you can:

- Explicitly "hard code" a specific value in a variable.
- Initialize a set of input variables by assigning values to all of the input variables.
- Assign a *default* value to a variable. That is, a value that is only assigned if the variable is null at run time.

- Assign a variable the value of another pipeline variable by referencing the variable. You might do this if you wanted to derive the default variable value from another variable in the pipeline at run time.
- Assign a variable the value of a global variable. You might do this for values that will change after you deploy a solution such as connection credentials. Instead of changing the value assigned to the pipeline variable in each service that establishes a connection, you change the value once in the global variable definition. For more information about global variables, see "Assigning Global Variables to Pipeline Variables" on page 274

By using to assign a value to a variable, you instruct Integration Server to write a specific value to that variable at run time. This action occurs just before the selected service is executed (if you assign the value to a variable in **Service In**) or immediately after the selected service is executed (if you assign the value to a variable in **Pipeline Out**).

Assigning a Value to a Pipeline Variable

You can assign values to variables that are in **Service In** or **Pipeline Out** when the variable is *not* linked or when the variable is only implicitly linked to another value in the pipeline.

You *cannot* assign values to:

- Variables that are explicitly linked to another value in the pipeline
- Variables that have been dropped from the pipeline
- Object variables constrained as a byte [ ]
- Unconstrained Objects (Objects of unknown type)
- An XML document variable or an XML document list variable

You cannot assign a value to a pipeline variable if the variable has a fixed null or default value assigned to it. Designer uses the symbol next to the variable icon to indicate that the variable has a fixed value that you cannot override by linking it to another variable.

To assign a value to a pipeline variable

1. In the flow service editor, select the INVOKE or MAP step containing the variable you want to alter.
2. In Pipeline view, select the variable to which you want to assign a value. The variable must be in either **Service In** or **Pipeline Out**.
3. Click on the Pipeline view toolbar.

   Designer displays the **Enter Input for** dialog box.
4. Assign values using the **Enter Input for** dialog box. For specific information for how to assign a value based on a variable's data type, see one of the following:

For this type of variable...	See...
String	"Specifying a Value for a String Variable" on page 408
String List	"Specifying Values for a String List Variable" on page 409
String Table	"Specifying Values for a String Table Variable" on page 411
Document	"Specifying Values for a Document Variable that Has Defined Content" on page 413 or "Specifying Values for a Document Variable with No Defined Content" on page 414
Document Reference	"Specifying Values for a Document Variable that Has Defined Content" on page 413 or "Specifying Values for a Document Variable with No Defined Content" on page 414
Document List	"Specifying Values for a Document List Variable" on page 416
Document Reference List	"Specifying Values for a Document List Variable" on page 416
Object	"Specifying a Value for an Object Variable" on page 418
Object List	"Specifying Values for an Object List Variable" on page 419

Notes:

- The **Include empty values for String Type** check box is disabled when assigning values to pipeline variables of type String, String List, String Table, Document, Object, and Object List. It is available only when assigning values to Document List variables. For more information, see "Specifying Values for a Document List Variable" on page 416.
- The check boxes next to each element in the tree are disabled when assigning values to pipeline variables of type String, String List, String Table, Document, Object, and Object List. The check box is only enabled for top-level Document variables within a Document List and is used along with the **Include empty values for String Type** check box. For more information, see "Specifying Values for a Document List Variable" on page 416.
- The **Perform pipeline variable substitution** check box indicates whether you want Integration Server to perform pipeline variable substitution at run time. To use a variable when assigning a String value, you type the name of the pipeline variable enclosed in % symbols (for example, `%Phone%`). If you specify a pipeline variable enclosed in % symbols for a String value, you must select the **Perform pipeline variable substitution** check box for the variable substitution to occur.
- The **Perform global variable substitution** check box indicates whether you want Integration Server to perform global variable substitution at run time. To use a global variable when assigning a String value, you type the name of the global variable enclosed in % symbols (for example, `%myFTPUsername%`). If you specify a global variable enclosed in % symbols for a String value, you must select the **Perform global variable substitution** check box for the variable substitution to occur.

- If a pipeline variable and global variable have the same name and you select both **Perform global variable substitution** and **Perform pipeline variable substitution**, Integration Server uses the value of the pipeline variable.
- The **Overwrite pipeline value** check box indicates whether you want the server to use the value you specify even when the variable has a pipeline value at run time.
  - Select the check box to have Integration Server always use the value you specify.
  - Clear the check box if you want Integration Server to use the value you specify only if the variable does not contain a value at run time.

Assigning Global Variables to Pipeline Variables

A global variable is a key/value pair that you define using Integration Server Administrator. You can use a global variable in a flow service by assigning the global variable to a variable in the pipeline (specifically a variable in **Service In** or **Pipeline Out**). At run time, Integration Server uses the value of the global variable as the value of the pipeline variable.

Using global variables makes it easy to change the value assigned to a pipeline variable at run time or after you deploy a solution to a different Integration Server. Instead of changing the hard-coded value of a pipeline variable, you change the value of the global variable. For example, you could create global variables for the connection parameters required by pub.client:ftp service. You might create global variables for the FTP server name, a user on the FTP server, and password for that user. In flow services that invoke pub.client:ftp you could assign the global variables to the *serverhost*, *username*, and *password* input parameters. After deploying the flow service to different servers, you may need to use different values for the *serverhost*, *username*, and *password* input parameters. You can use Integration Server Administrator to change the value of the global variables on the target servers. This is more efficient than editing the services to change the hard-coded values.

Keep the following points in mind when using global variables in flow services:

- You can use global variables to specify values for variables of type String, String List, and String Table only.
- To specify a global variable as the value for a pipeline variable, you enclose the name of the global variable in % symbols (for example, `%myFTPServer%`).
- You must select the **Perform global variable substitution** check box for the variable substitution to occur at run time.
- If the specified global variable has the same name as a pipeline variable name you select the **Perform global variable substitution** check box *and* the **Perform pipeline variable substitution** check box, Integration Server uses the value of the pipeline variable at run time.
- If the global variable that you specified for performing a variable substitution is not defined in Integration Server, at run time Integration Server throws an exception and service execution fails.

- You can mix literal and global variables. For example, if you specify `(%areaCode%) %Phone%`, the resulting String would be formatted to include the parentheses and space. If you specify `%firstName% %initial%. %lastName%`, the period and spacing would be included in the value.

For more information about defining global variables, see *webMethods Integration Server Administrator's Guide*.

Copying Assigned Values Between Pipeline Variables

You can copy the value assigned to a variable by copying the icon next to the variable. You can assign this value to other variables of the same data type in **Service In** or **Pipeline Out** by pasting the icon.

When you copy assigned values from one pipeline variable to another, keep the following points in mind:

- You can only copy and paste set values between variables of the same data type. For example, you can only copy the set value assigned to a String variable to another String variable.
- You can only copy and paste set values between variables if the target variable has the same structure as the source variable or has no defined structure. For example, you can copy the set value of a String List variable with length 3 to another String List variable only if the target String List also has length 3 or has an undefined length (no defined structure).
- If you are copying a set value between Document variables, the source Document variable and the target Document variable must have the same structure or the target Document variable must have no structure defined. For example, if the source Document variable contains three String variables named *city*, *state*, and *zip* as children, the target Document variable must have three String variables named *city*, *state*, and *zip* as children.
- You cannot copy an assigned value to a pipeline variable if the variable has a fixed null or default value assigned to it. Designer uses the symbol next to the variable icon to indicate that the variable has a fixed value that you cannot override by linking it to another variable.

To copy a set value

1. In the flow service editor, select the INVOKE or MAP step containing the variable with the value you want to copy and paste.
2. In the Pipeline view, select the assigned value icon that you want to copy.
3. Right-click and select **Copy**.
4. Select the variable to which you want to assign the copied value, right-click and select **Paste**.

Dropping Variables from the Pipeline

You can remove a variable from **Pipeline In** or **Pipeline Out** by dropping the variable. You can drop variables to eliminate pipeline variables that are not used by subsequent services in a flow.

Dropping unneeded variables reduces the size of the pipeline at run time and reduces the length and complexity of the **Pipeline In** and **Pipeline Out** displays, which can make the Pipeline view much easier to use when you are working with a complex flow.

Important: Software AG recommends that you drop variables containing sensitive data from the pipeline once the data is no longer needed by the flow service. Sensitive data includes, but is not limited to, passwords, private keys, and pass phrases. The pipeline stores data in clear text. Because the entire output pipeline is returned to the calling client, avoid leaving variables containing sensitive data in the pipeline. This prevents the service from returning sensitive data to the calling client.

Keep the following points in mind when dropping variables from the pipeline:

- You can only drop variables from **Pipeline In** and **Pipeline Out**. In a MAP step, you can only drop variables from **Pipeline In**.
- Once you drop a variable from the pipeline, it is no longer available to subsequent services in the flow. Do not drop a variable unless you are sure the variable is not used by services invoked after the point where you drop it.
- At run time, Integration Server removes a dropped variable from the pipeline just before it executes the selected service (if you drop a variable in **Pipeline In**) or immediately after it executes the selected service (if you drop a variable in **Pipeline Out**).
- If you drop a linked variable from **Pipeline In**, Integration Server executes the link *before* it drops the variable. However, Integration Server server does not link a null value to the destination variable.
- You cannot drop a pipeline variable if the variable has a fixed null or default value assigned. Designer uses the symbol next to the variable icon to indicate that the variable has a fixed value that you cannot override by linking it to another variable.
- You cannot drop a pipeline variable in a child flow service if the variable exists in the parent flow service. That is, a child flow service cannot drop an “upstream” variable.

To drop a variable from the pipeline

1. In the flow service editor, select the INVOKE or MAP step whose pipeline variables you want to drop.
2. In the Pipeline view, select the variable that you want to drop.
3. Click on the toolbar.

Adding Variables to the Pipeline

In the Pipeline view, you can add variables that were not declared as input or output parameters for the flow service itself or any of its constituent services. You can add variables that were omitted from a service’s input or output parameters or create temporary variables for use within the flow.

For example, you might attach a variable to each of the children in a BRANCH step to mark the path taken by the service at run time.

Variables that you add to the pipeline can be used just like any declared variable in the flow.

Keep the following points in mind when adding variable to the pipeline:

- If you create a new variable in a flow, you must *immediately* do one of the following:
  - Link a variable to it
  - Assign a value to it
  - Drop it

  If you do not take one of these steps, Designer automatically clears it from the pipeline.
- You might want to drop a variable immediately after adding it if a service produces a variable that is not declared in the service input or output parameters. The variable will not appear in the Pipeline view if it is not an input or output parameter. By adding and then immediately dropping the variable, you can delete the variable if it does exist in the pipeline.

To add a variable to the pipeline

1. In the flow service editor, select the INVOKE or MAP step that represents the stage of the pipeline at which you want to add a new variable.
2. Do one of the following in the Pipeline view:
   - Select the point in the pipeline where you want to add the new variable (**Pipeline In**, **Service In**, **Service Out**, or **Pipeline Out**). Click and select the type of variable that you want to create.
   - In the Palette view that is part of Pipeline view, under Variables, select the variable that you want to add and then select the point in the pipeline where you want to add it. The Palette view is located within the Pipeline view. Click to show the Palette view. Click to hide the Palette view.
3. Type a name for the variable and press ENTER.
4. With the variable selected, set variable properties and apply constraints using the Properties view.
5. If the variable is a Document or a Document List, add more variables to define its contents. Then use to indent each member variable beneath the Document or Document List variable.
6. Do one of the following with the new variable:
   - Link the variable to another variable.
   - Assign a value to the variable using on the Pipeline view toolbar.

- Drop the variable.

Working with Transformers

Transformers are services that are inserted into and executed within a MAP step. You can use any service as a transformer. This includes any Java, C, or flow service that you create and any built-in services in WmPublic, such as the pub.date.getCurrentDateString and pub.string.concat. Additionally, you can insert multiple transformers into a single MAP step. By using multiple transformers, you can perform multiple data transformations on the pipeline contents in a single flow step.

Transformers act as collection of INVOKE steps embedded in a single MAP step. However, transformers in a MAP step are independent of each other, do not execute in a specific order, and might not execute in the same order each time the MAP step runs. Consequently, the output of one transformer cannot be used as the input to another transformer. These characteristics make transformers different than a set of INVOKE steps that execute sequentially in a flow service. Because transformers are contained within a MAP step, they do not appear as a separate flow step in the editor.

The purpose of transformers is to accomplish multiple data transformations on the pipeline data in a single step as opposed to using succession of INVOKE steps. As a result, transformers are well suited to use when mapping data from one document format to another. When mapping data between formats, you often need to perform several name, structure, and value transformations. With the use of transformers, the flow service in which you map data between formats could potentially consist of a single MAP step in where transformers and links between variables handle all of the data transformations. This provides a single view of document-to-document mapping. For example, you could create a flow service that uses transformers to convert data between document formats (such as an IDOC to an XML document or RosettaNet PIP to a proprietary format). Each time you need to convert between the specific document formats, you could invoke the mapping service.

Note:
When determining which services to use as transformers, Software AG recommends avoiding services that are subject to transient failures, such as a connection failure, as these services may be difficult to debug when used as a transformer.

Using Built-In Services as Transformers

Integration Server provides several built-in services specifically designed to translate values between formats. These services can be found in the following folders in the WmPublic package:

This folder…	Contains services to…
pub.date	Transform time and date information from one format to another.
pub.list	Transform a String List to a Document List (IData[ ] object) and append items to a Document List (IData[ ] object) or a String List.
pub.math	Perform simple arithmetic calculations (add, subtract, multiply, and divide) on integers and decimals contained in String variables.

This folder…	Contains services to…
pub.string	Transform String values in various ways (for example, pad, substring, concat, replace through a lookup table).

For more information about built-in services, see the *webMethods Integration Server Built-In Services Reference*.

Inserting a Transformer

When inserting transformers, keep the following points in mind:

- Transformers can be inserted in a MAP step only.
- Any service can be used as a transformer, including flow services, C services, and Java services.
- The transformers in a single MAP step operate on the same set of pipeline data.
- Transformers in a MAP step are independent of each other and do not execute in a specific order. As a result, the output of one transformer cannot be used as the input of another transformer in the same MAP step.
- Software AG recommends avoiding the use of a service as a transformer if the service is subject to transient failures, such as a connection failures, as these services might be hard to debug when used as a transformer.

To insert a transformer

1. In the flow service editor, select the MAP step in which you want to insert a transformer.
2. In the Pipeline view, do one of the following:
   - Click the ▾ button adjacent to on the Pipeline view toolbar and select the service you want to use as a transformer. If the service you want to insert does not appear in the list, click **Browse** to select a service on Integration Server.
   - In the Palette view that is located within the Pipeline view, select the folder containing the service you want to add as a transformer. Select the service and click in the **Transformers** area of Pipeline view.
   - In Package Navigator view, select the service you want to use as a transformer and drag it to the **Transformers** area of Pipeline view.
3. To set properties for the transformer, select it and then specify the following information in the Properties view:

For this property...	Specify...
Service	The fully qualified name of the service that will be invoked at run time as a transformer. When you insert a transformer, Designer automatically assigns the name of that service to the service property. If you want to change the service that is invoked by a transformer, specify the service's fully qualified name in the *folderName:serviceName* format or click ... to select a service from a list.
Validate input	Whether Integration Server validates the input to the transformer against the signature of the service. Select **True** to validate the input of the transformer, otherwise select **False**.
Validate output	Whether Integration Server validates the output of the transformer against the signature of the service. Select **True** to validate the output of the transformer, otherwise select **False**.

4. Link pipeline variables to the transformer variables. See " Linking Variables to a Transformer" on page 280.

Linking Variables to a Transformer

When you map data to and from a transformer, you create links between the pipeline variables and the transformer. Keep the following points in mind when you create links between pipeline and transformer variables:

- You must explicitly link pipeline variables to the input and output variables of a transformer. Designer does not perform any implicit linking with transformers. Even if the pipeline variables have the same name and data type as the transformer variables, no implicit linking occurs.
- Designer does not automatically add the output of a transformer to the pipeline. If you want the output of a transformer to appear in the pipeline, you need to explicitly link the output variable to a **Pipeline Out** variable.
- If you do not link any output variables or the transformer does not have any declared output variables, the transformer service will not run.
- You can link a transformer output variable to more than one **Pipeline Out** variable.
- You can assign a value to a transformer input value using on the Pipeline view toolbar.
- To prevent the Pipeline view from becoming cluttered, the Pipeline view may not display all the links between the transformer and the pipeline variables. To view all the links, double-click the transformer or click next to the transformer name.

Use the following procedure to link pipeline and transformer variables when the transformer is not expanded. If the transformer is expanded (that is, you can see all of the input and output variables for the transformer), you link variables just as you would for an INVOKE step.

To create a link between a pipeline variable and a transformer

1. To create a link between a **Pipeline In** variable and a transformer variable, do the following:

   a. In **Pipeline In**, select the variable you want to use as input to the transformer and drag your mouse to the collapsed transformer. Designer displays the **Link** dialog box.

   b. In the **Link To** list, select the transformer variable to which you want to link the **Pipeline In** variable.

      In the **Link To**list, Designer displays the phrase "has already been chosen" next to variables that are already linked to other variables transformer.

2. To create a link between a transformer output variable and a **Pipeline Out** variable, do the following:

   a. Select the collapsed transformer and drag your mouse to the variable in **Pipeline Out** to which you want to link the transformer variable. Designer displays the **Link** dialog box.

   b. In the **Link From** list, select the transformer variable that you want to link to the selected **Pipeline Out** variable.

Transformers and Array Variables

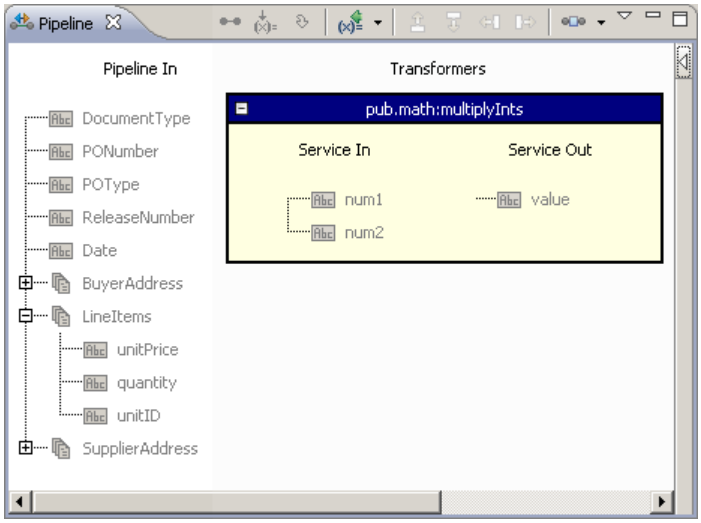
When creating links between pipeline variables and transformers, dimensional differences between the source and target variables may cause an exception. If the target variable dimensionality is greater than the source variable dimensionality, an exception will not be thrown. However, if the source variable dimensionality is greater than the target variable dimensionality, Integration Server throws an exception.

Dimensionality refers to the number of arrays to which a variable belongs. For example, the dimensionality of a single String is 0, that of a single String List or Document List is 1, and that of a single String Table is 2. A String that is a child of a Document List has a dimensionality of 1. A String List that is a child of a Document List has a dimensionality of 2.

Example of Dimensionality Mismatch

In the following example, the *unitPrice* variable cannot be linked to *num1* because the *unitPrice* variable has a dimensionality of 1 (String (0) + Document List (1) = 1) and *num1* has a dimension of 0.

unitPrice cannot be linked to num1 because of dimensionality differences



To solve this, you can either:

- Change the service invoked by the transformer to accept arrays as data, or
- Create a flow service in which a LOOP step loops over the array variable. Then, (in the same flow service) invoke the service you originally wanted to use as a transformer, and make that INVOKE step a child of the LOOP. Finally, insert the resulting flow service as a transformer in the MAP.

Of the two options, changing the service to accept arrays as data results in faster execution of flow services.

Validating Input and Output for Transformers

As with any service you insert using an INVOKE, you can validate the inputs and outputs of the transformer service before and/or after it executes. To indicate that you want to validate a transformer's inputs and outputs, you change the properties of the transformer. You do not have to use validation for all of the transformers you insert into a MAP step.

When Integration Server validates a transformer's inputs and outputs at run time, Integration Server validates the transformer input and output values against the signature of invoked service. Variables in the service signature may specify content or structural constraints.

Note:
If the **Validate input** and/or **Validate output** check boxes are selected on the Input/Output tab of the service acting as a transformer, Integration Server automatically validates the input and/or output for the service every time the service executes. If you set up validation via the properties for a transformer when it is already set up for validation via the service's Input/Output tab, Integration Server performs validation twice. This can slow down the execution of a transformer and, ultimately, the flow service.

To specify input/output validation for a transformer

1. In the flow service editor, select the MAP step containing the transformer you want to validate.

2. In the Pipeline view, under **Transformers**, select the transformer for which you want to specifying input/output validation.

3. In the Properties view, do the following:

   - If you want Integration Server to perform input validation, set the **Validate input** property to **True**.
   - If you want Integration Server to perform output validation, set the **Validate output** property to **True**.

Copying Transformers

You may want to use the same transformer more than once in a MAP step. For example, you might want to convert all the dates in a purchase order to the same format. Instead of inserting the service repeatedly, you can copy and paste the transformer service.

- You can copy transformers between MAP steps in the same flow or MAP steps in different flow services.
- You can copy multiple transformers at a time.
- Copying a transformer does *not* copy the links between transformer variables and pipeline variables or any values you might have assigned to transformer variables using .

To copy a transformer

1. In the flow service editor, select the MAP step containing the transformer service you want to copy.

2. In the Pipeline view, under **Transformers**, select the transformer that you want to copy. Right-click and select **Copy**.

3. Do one of the following:

   - To paste the transformer in the same MAP step, right- click anywhere under **Transformers** and select **Paste**.
   - To paste the transformer in another MAP step, select that MAP step. In Pipeline view, right- click anywhere under **Transformers** and select **Paste**.

4. Link the input and output variables of the transformer. See " Linking Variables to a Transformer" on page 280.

Renaming Transformers

If Integration Server displays the message "Transformer not found" when you try to expand a transformer or when you point the mouse to the transformer, then the service referenced by the transformer has been renamed, moved, or deleted. You need to change the **Service** property of the transformer so that the transformer points to the moved, or renamed service.

If the service referenced by the transformer has been deleted, you may want to delete the transformer.

Tip:
You can enable safeguards so that you do not inadvertently affect or break other services when you move, rename, or delete a service. For more information, see "Configuring Dependency Checking for Elements" on page 45.

To rename a transformer

1. Use Package Navigator view to determine the new name or location of the service called by the transformer.

2. Open the flow service containing the transformer you want to rename.

3. In the flow service editor, select the MAP step containing the transformer. Then, in Pipeline view, select the transformer you want to rename.

4. In the **Service** property in the Properties view, delete the old name and type in the service's new fully qualified name in the *folderName:serviceName* format, or click ... to select a service from a list.

Debugging Transformers

When you debug a flow service, you can use the following debugging techniques with transformers:

- Step into a MAP step and step through the execution of each transformer. For more information about stepping into and out of a MAP Step, see "Stepping Into and Out of a MAP Step" on page 437.
- Set a breakpoint on a transformer so that service execution stops when the transformer is encountered. For more information about setting breakpoints, see "Setting and Removing Breakpoints on Flow Step" on page 441.
- Disable a transformer so that it does not execute at run time. For more information about disabling transformers, see "Disabling and Enabling Flow Steps and Transformers" on page 442.

Testing Flow Steps Before Running a Flow Service

As you construct a flow service, you can separately test the INVOKE or MAP steps that you plan to use in the flow service. For each step, you can provide relevant inputs, run the step, and check the output before running the entire flow service. This approach helps in early validation of a MAP or INVOKE step before it passes data to the next step in the flow.

You need to use the Data Mapper view in Designer for testing the flow steps.

Opening the Data Mapper View

To open the Data Mapper View

1. In the Service Development perspective of Designer, select **Window > Show View > Other**.
2. In the Show View dialog box, select **Software AG Service Development > Data Mapper**.
3. Click **OK**.

   Designer displays the Data Mapper view.

Testing a Flow Step in the Data Mapper View

To test a flow step in the Data Mapper View

1. In the Package Navigator view, open the flow service for which you want to insert the required INVOKE or MAP steps.
2. Add the required flow steps, as explained in "Inserting Flow Steps" on page 213.
3. Select the flow step to test, and switch to the Data Mapper view.
4. In the Mapping tab of the Data Mapper view, define the required pipeline variables and transformers, and create links.

   You can perform these operations in the Mapping tab as you would in the Pipeline view. For more information, see the following sections:

   - "About Linking Variables" on page 256
   - "Adding Variables to the Pipeline" on page 276
   - "Working with Transformers" on page 278
5. In the Testing tab of the Data Mapper view, specify values for the input variables of the flow step in the Input Value Creation area.

You can use either of the following approaches to specify the values:

- In the Value column of the Input Value Creation area, type the required values against the corresponding variable names listed in the Name column.
- To load input values, which match the structure of the flow service's input signature, from a file; click **Load**, and select the appropriate file.
- To load input values from a file, and replace the flow service's input signature with structure and data types from the file; click **Load and Replace**, and select the appropriate file.

Note:
For more information about loading input values from a file, see "Loading Input Values" on page 420.

6. If you want to save the specified input values for later use, click **Save**.

   Otherwise, directly go to Step 7.

7. To test the flow step based on the specified input values, click .

   Designer displays the results of the test in the Test Outcome area of the Data Mapper view.

   Depending on the results of a test execution, the Test Outcome area displays information in the following tabs:

   - The **Pipeline** tab displays the contents of the pipeline when the test finishes executing.
   - The **Messages** tab displays messages from Designer about the test and any exception thrown during the execution.

Mapping Using ForEach

To simplify array mapping, Designer provides **ForEach** mapping in the pipeline view of a MAP step. **ForEach** mapping provides following capabilities:

- Reduces multiple flow steps: Nested array mapping involves multiple flow steps. Now, using **ForEach** mapping, you can easily create nested mapping in a single MAP step.
- Copy modes: When an input array is mapped to an output array, the value of output array is merged with input array. Using **ForEach** mapping, you can now merge, overwrite, and append the values to output array.
- Multiple transformer invocation: **ForEach** mapping allows you to directly apply multiple transformers on input array elements while mapping.

Creating a ForEach Mapping

Consider the following when you build a **ForEach** mapping:

- You can only map an input array to an output array of the same data type.

- **ForEach** mapping iterates over an input array and executes the defined mappings.
- You can perform all the mapping tasks, transformations, and create nested array mapping in a **ForEach** mapping.
- You can map a single input array to multiple output arrays and multiple input arrays to a single output array.

To build a ForEach mapping

1. In the flow service editor, select the MAP step.
2. Open the Pipeline view.
3. In **Pipeline In**, click the pipeline array variable you want to use as the source variable and in **Pipeline Out**, click the pipeline array variable you want to use as the target variable.
4. Click .

   The **ForEach** mapping is established between the source and target arrays.
5. On the Pipeline view of the **ForEach** mapping, click .

   The Pipeline view of the selected **ForEach** mapping appears.
6. In the Pipeline view **ForEach** mapping, add the required mapping tasks, transformations, or nested mapping.
7. Use to go to the Pipeline view of the parent **ForEach**, in case of nested **ForEach** mapping.
8. Use to close the **ForEach** mapping Pipeline view.
9. Click **File > Save** .

Specifying ForEach Mapping Properties

Following are the properties of **ForEach** mapping:

- **Filter Input**: Specifies the matching condition for filtering the input array elements. At run time, Integration Server evaluates the condition for each input array element and if the condition evaluates to true, the mappings defined in the **ForEach** mapping pipeline are executed.

  Note:

For information about the syntax used in conditions, see "Conditional Expressions" on page 1113.

- **Copy Mode**: Specifies how to copy the elements from input to output array. Following copy modes are supported:
  - Append: Appends the output array.
  - Merge: Updates the existing output array with source values. This is the default mode.
  - Overwrite: Overwrites the output array.
- **Elements Selection**: Specifies the indexes of the input array that participate in the **ForEach** mapping. The first index element is 0. You can specify multiple cardinalities using the " , " delimiter. Consider the following example:

If you specify an index value...	Then ForEach processes...
3	only the fourth element of the input array.
1,5	only the second and the sixth elements of the input array.
1-6,9,10	all second to seventh, tenth, and eleventh elements of the input array.
Empty	all the elements of the input array.

Note:
You should specify the indexes in the increasing order.

Note:
When you specify both **Elements Selection** and **Filter Input**, then **ForEach** first selects elements based on the **Elements Selection** and then applies the **Filter Input** conditions on selected elements.

To specify ForEach mapping properties

1. Create a **ForEach** mapping between the array variables. See "Creating a ForEach Mapping" on page 286.
2. On the **ForEach** mapping, click .

   The **ForEach Settings** dialog box appears.
3. On the **Filter Input** screen, enter the filter condition for the **ForEach** mapping and click **Next**.
4. In the **Copy Mode** screen, select the required copy action and click **Next**.

5. In the **Elements Selection** screen, enter the indexes of the input array.

6. Click **Finish**.

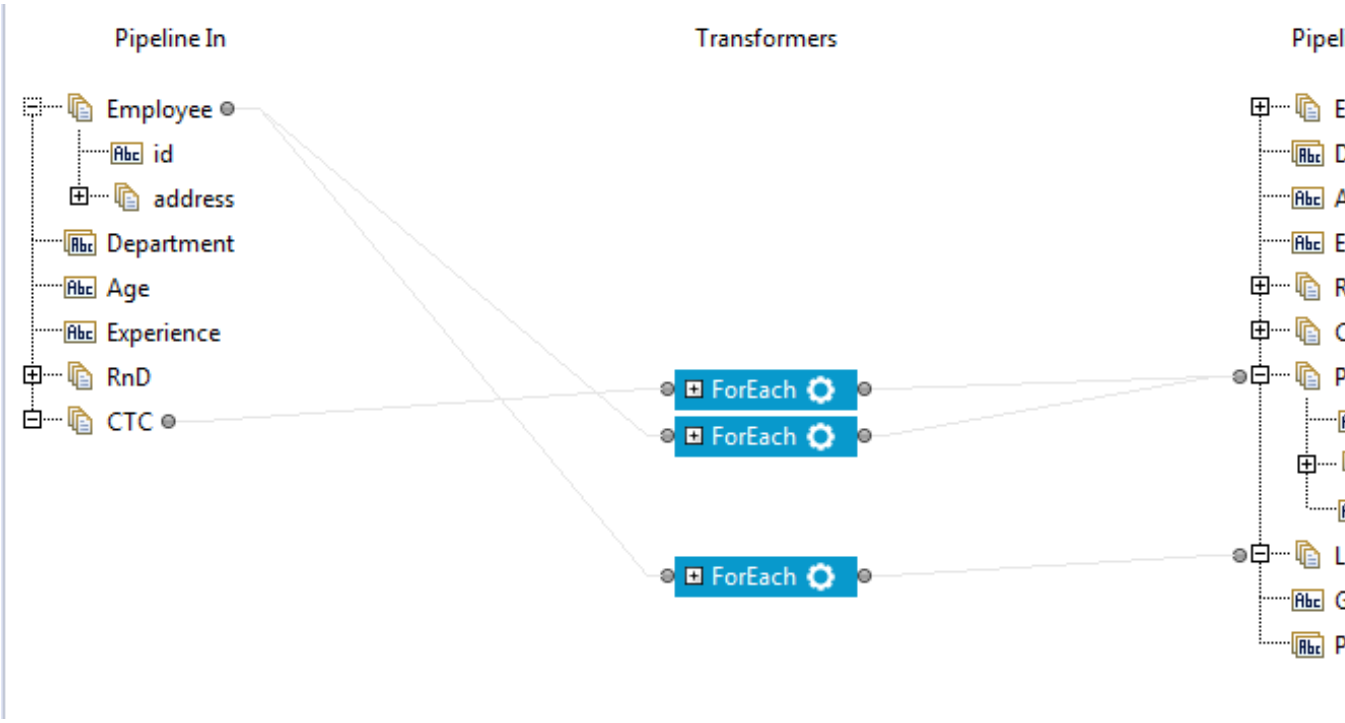
   A **ForEach** mapping is established between the selected variables.

ForEach Mapping Rules

The following tables identify the default **ForEach** mapping rules for array variables.

The example below illustrates a **ForEach** mapping between Employee (input array) and Person (output array).

1. In a**ForEach** mapping, you cannot link the individual array elements in a Pipeline view. To do this, you must expand the **ForEach**mapping. Also, you can build a**ForEach** mapping from a nested input array, only if parent array is part of **ForEach** mapping.



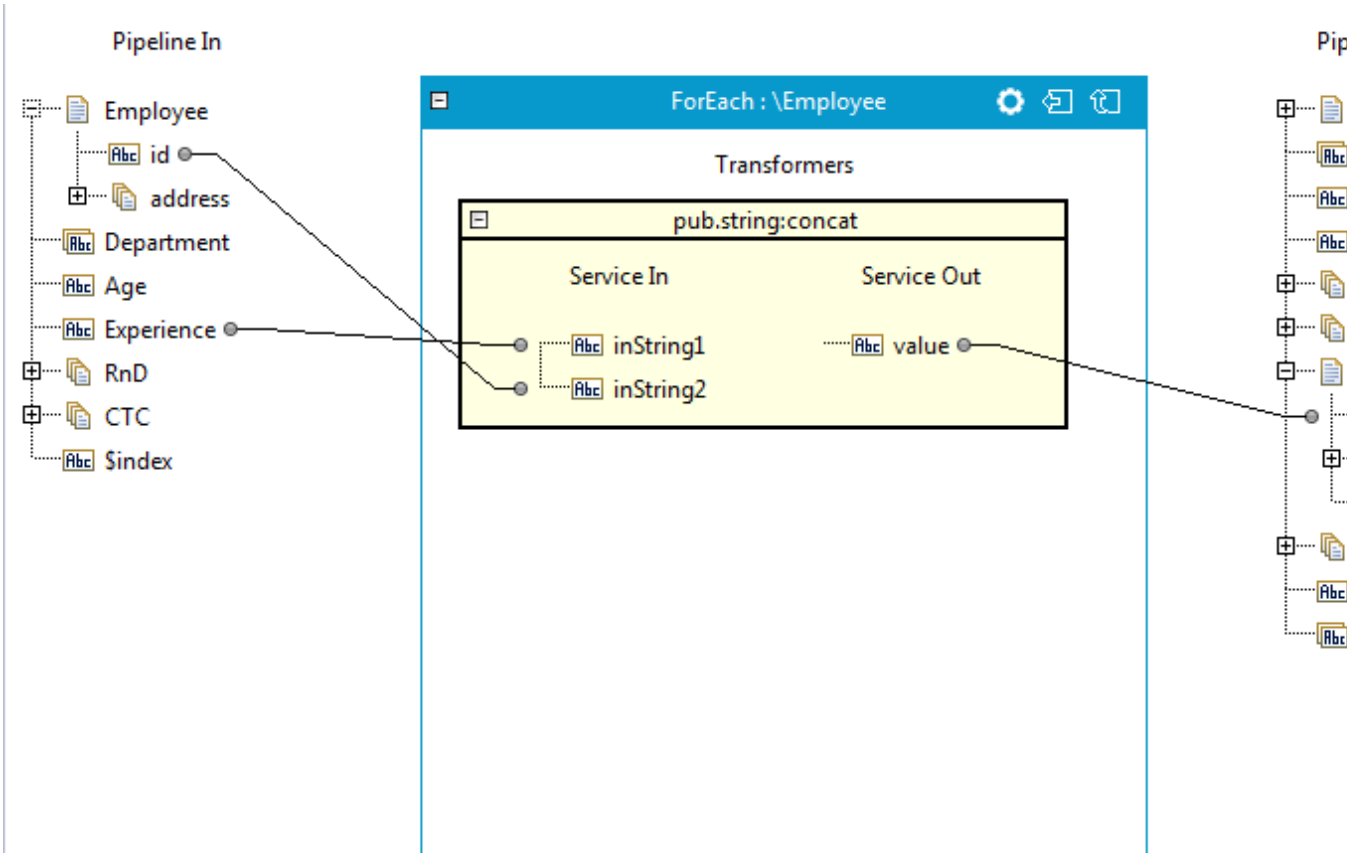
   Following table identifies the possible cases in Pipeline view.

From...	To...	Then…
/Employee/id	/Person/name	This is an invalid link. Employee and Person arrays are already part of **ForEach** mapping. To create this link, go to the Pipeline view of the **ForEach** mapping.

From...	To...	Then…
Age	Person/age	This is an invalid link. Person is part of current **ForEach** mapping but Age is not. To create this link, go to the Pipeline view of the **ForEach** mapping.
Employee	Location	This is a valid **ForEach** mapping. Single input array (Employee) to multiple output array (Location and Person).
CTC	Person	This is a valid **ForEach** mapping. Multiple input arrays (CTC and Employee) to single output array (Person).
Employee	Products	This is an invalid **ForEach** mapping. Employee and Products are of different data types.
Employee/address	Person/city	This is an invalid **ForEach** mapping. Employee and Person are already part of current **ForEach** mapping. To establish this nested **ForEach** mapping, go to the Pipeline view of the **ForEach** mapping.
RnD/products	Products	This is an invalid **ForEach** mapping. RnD is not part of any **ForEach** mapping. To establish this nested **ForEach** mapping, add a **ForEach** mapping for RnD in the Pipeline view and then go to the Pipeline view of the **ForEach** mapping.

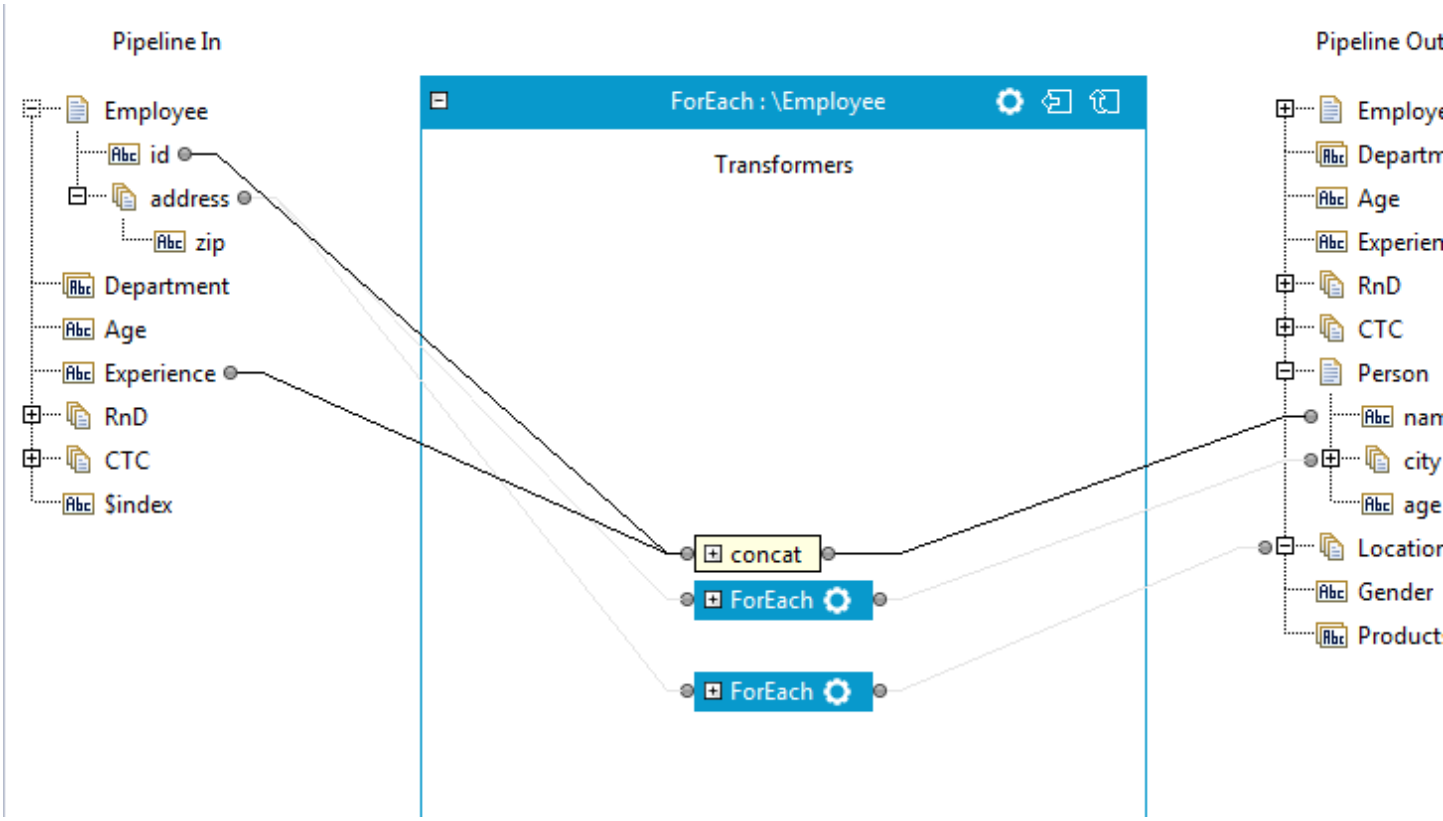
2. In the Pipeline view of the **ForEach** mapping, a link is allowed only if the source variable, target variable, or both the variables are part of the current **ForEach** mapping array elements.

   Following table identifies the possible cases in the Pipeline view of the **ForEach** mapping.



From...	To...	Then…
/Employee/id	Age	This is a valid link. Employee/id is part of the current **ForEach** mapping.
/Employee/id	Person/name	This is a valid link. Employee and Person arrays are part of **ForEach** mapping.
Experience	Person/age	This is a valid link. Person/age is part of **ForEach** mapping.
RnD/products	Products	This is an invalid link. RnD is not part of parent**ForEach** mapping (Employee>Person). To create this link, go back to Pipeline view of MAP step.

3. In the Pipeline view of the **ForEach** mapping, you can build a nested **ForEach** mapping only from those input array elements that are part of the parent **ForEach** mapping.



Following table identifies the possible cases:

From...	To...	Then…
/Employee/address	Person/city	This is a valid nested **ForEach** mapping. Employee/address is an element of parent **ForEach** mapping.
/Employee/address	Location	This is a valid nested **ForEach** mapping. Employee/address is an element of parent **ForEach** mapping.
RnD	Department	This is an invalid nested **ForEach** mapping. RnD is not an element of parent **ForEach** mapping (Employee).

13 Performing Data Validation

- Blueprints or Models Against which Data is Validated 294
- Performing Input/Output Validation 295
- Performing Pipeline Validation 297
- Performing Document Validation 297
- Performing XML Validation in Integration Server 298
- Performing Validation from within a Java Service 299
- Validation Errors 300
- Validation Exceptions 300
- Preventing Running Out of Memory Error During Validation 301

Data validation is the process of verifying that run-time data conforms to a predefined structure and format. Data validation also verifies that run-time data is a specific data type and falls within a defined range of values.

By performing data validation, you can make sure that:

- The pipeline, a document (IData object), or an XML document contains the data needed to execute subsequent services. For example, if a service processes a purchase order, you might want to verify that the purchase order contains a customer name and address.
- The data is in the structure expected by subsequent services. For example, a service that processes a purchase order might expect the customer address to be a document field with the following fields: `name`, `address`, `city`, `state`, and `zip`.
- Data is of the type and within a value range expected by a service. For example, if a service processes a purchase order, you might want to make sure that the purchase order does not contain a negative quantity of an item (such as -5 shirts).

By using the data validation capabilities built into Integration Server, you can decide whether or not to execute a service based on the validity of data. The validation capabilities can also eliminate extra validation code from your services.

Blueprints or Models Against which Data is Validated

During validation, run-time data is compared to a blueprint or model. The blueprint or model is a formal description of the structure and the allowable content for the data. The blueprint identifies structural and content constraints for the data being validated. The validation engine in Integration Server considers the data to be valid when it conforms to the constraints specified in the blueprint. A blueprint can be an IS schema, an IS document type, or a set of input and output parameters.

The blueprint used to validate data depends on the type of validation you are performing. In Integration Server, you can perform the following types of validation:

- **Input/Output validation**. The validation engine in Integration Server validates the input and/or output of a service against the signature of the service.
- **Document validation.** The validation engine in Integration Server validates the structure and content of a document (IData object) in the pipeline against an IS document type.
- **Pipeline validation.** The validation engine in Integration Server validates the structure and content of the pipeline against an IS document type.
- **XML validation.** The validation engine in Integration Server validates the structure and content of an XML document against an IS schema or IS document type.

For input/output, document, and pipeline validation, the blueprint used for data validation requires *constraints* to be applied to its variables. Constraints are the restrictions on the structure or content of variables. Structural constraints specify the existence and structure of variables at run time. Content constraints describe the data type for a variable and the possible values for the variable at run time. For more information see, "About Variable Constraints" on page 607.

Performing Input/Output Validation

In input/output validation, the validation engine in Integration Server validates the inputs and/or outputs of a service against the declared input and output parameters of the service. If you specify that you want to validate the inputs to the service, the validation engine validates the service input values immediately before the service executes. If you specify that you want to validate the outputs of the service, the validation engine validates the service output values immediately after the service executes. An input or output value is invalid if it does not conform to the constraints applied to the input or output parameter.

For input/output validation, the service's declared input and output parameters act as the blueprint or model against which input/output values are validated. To effectively use the input and output parameters as the blueprint for validation, you need to apply constraints to the parameters. For information about applying constraints to variables, see "About Variable Constraints" on page 607. For information about declaring service input and output parameters, see "Declaring Input and Output Parameters" on page 162.

Note:
The declared input and output parameters for a service are sometimes called the signature of the service.

You can specify that you want to perform input/output validation for a service in the following ways:

- **Input/Output tab.** Set properties on the **Input/Output** tab to instruct the validation engine in Integration Server to validate the inputs and/or outputs of the service every time the service executes. If a client calls the service and the inputs are invalid, the service fails and does not execute.
- **INVOKE step properties.** Set up input/output validation via the INVOKE step properties to instruct the validation engine to validate the service input and/or output only when it is called from within another flow service. At run time, if the inputs and/or outputs of the service are invalid, the INVOKE flow step that calls the service fails.

To determine which method to use, decide whether or not you want the service input and output values validated every time the service runs. If you want to validate the input and output values every time the service runs, specify validation via the **Input/Output** tab. For example, if your service requires certain input to exist or fall within a specified range of values, you might want the pipeline validated every time the service runs.

If the input and/or output values do not need to be validated every time the service executes, set up validation via the INVOKE step properties. Specifying input/output validation via the INVOKE step properties allows you to decide on a case-by-case basis whether you want validation performed

Note:
If you specify input/output validation via the INVOKE step and an input or output value is invalid, the service itself does not actually fail. The validation engine validates input values before Integration Server executes the service. If the service input is not valid, the INVOKE flow step for the service fails. Similarly, the validation engine validates output values after Integration Server executes the service. If the service output is not valid, the INVOKE flow step for the

service fails. Whether or not the entire flow service fails when an individual flow step fails depends on the exit conditions for the service. For information about specifying exit conditions, see "Using SEQUENCE to Specify an Exit Condition" on page 232.

Specifying Input/Output Validation via the Input/Output Tab

You can specify that you want the inputs and/or outputs of a service validated *every* time it executes by setting properties on the **Input/Output** tab of the service. Every time the service executes, the validation engine validates the input and/or output values of the service against the input and output parameters declared for the service.

To set up input and output validation via the Input/Output tab

1. In the Package Navigator view, open the service for which you want to validate input and/or output.
2. Select the **Input/Output** tab.
3. If you want to validate the input of the service every time the service executes, select the **Validate input** check box.
4. If you want to validate the output of the service every time the service executes, select the **Validate output** check box.
5. Select **File > Save**.

Specifying Input/Output Validation via the INVOKE Step

You can specify input/output validation by setting properties for the INVOKE step that calls the service. Each time you insert a service into a flow, you can decide whether you want the validation engine to validate service inputs and/or outputs at run time.

To set up input and output validation via the INVOKE step

1. In the flow service editor, select the INVOKE step for the service for which you want to validate input and/or output values.
2. In the Properties view under **General**, do the following:
   - To validate input to the service, select **True** for **Validate input**.
   - To validate the output of the service, select **True** for **Validate output**.
3. Select **File > Save**.

Important:

Keep in mind that the **Validate input** and **Validate output** properties are independent of any validation settings that you might have already set in the service. If you select the **Validate input** and/or **Validate output** check boxes on the **Input/Output** tab of the invoked service, Integration Server performs input/output validation every time the service executes. If you also specify input/output validation via the INVOKE step, duplicate validation will result, possibly slowing down the execution of the service.

Performing Pipeline Validation

Pipeline validation is the process of verifying the contents of the pipeline against an IS document type. By validating pipeline data, you can:

- Ensure a higher degree of accuracy for pipeline content.
- Execute or not execute a service based on the validity of the pipeline data.
- Eliminate extra validation code in your service.

In pipeline validation, an IS document type is the blueprint or model against which you validate the pipeline. The constraints applied to the variables in the IS document type determine what can and cannot be included in the pipeline. For more information about applying constraints to variables, see "About Variable Constraints" on page 607.

To validate the pipeline, invoke the built-in service pub.schema:validatePipeline. This service instructs the validation engine to compare the pipeline contents against a specified IS document type. The pipeline is valid when it conforms to the structural and content constraints applied to the IS document type. The pub.schema:validatePipeline service returns a string that indicates whether validation was successful and an IData array (*errors* variable) that contains any validation errors. For more information about the pub.schema:validatePipeline service, see the *webMethods Integration Server Built-In Services Reference*.

If you want to validate only String, String list, or String table variables in the pipeline, use the pub.schema:validatePipeline service. Define an IS document type that contains the variables you want to validate and apply constraints to the variables. Then use this IS document type as the blueprint for the pub.schema:validatePipeline service. Integration Server validates only the variables in the IS document type. The validation engine ignores other variables that exist in the pipeline at run time. (An IS document type implicitly allows unspecified variables to exist at run time.)

Performing Document Validation

You might want to validate an individual document (IData object) in the pipeline instead of the entire pipeline. Use the pub.schema:validate service to validate a single document (IData object) in the pipeline against an IS document type.

For example, suppose that you invoke the pub.client.ldap:search service in a flow to retrieve an IData object from an LDAP directory service. If you want to validate that object before you use it in other services, invoke the pub.schema:validate service after retrieving the object. As another example, you might want to validate an XML document that has been converted to a document (IData object). You would use the pub.schema:validate service to validate the resulting document (IData object) against an IS document type.

The pub.schema:validate service considers a document (IData object) to be valid when it complies with the structural and content constraints described in the IS document type it is validated against. This service returns a string that indicates whether validation was successful and an IData array that contains any validation errors. When you insert the pub.schema:validate service into a flow service, you can specify the maximum number of errors that the service can collect. You can also specify whether the pub.schema:validate service should fail if the document (IData object) is invalid.

For more information about the pub.schema:validate service, see the *webMethods Integration Server Built-In Services Reference*

Note:
The validation engine in Integration Server can perform document (IData object) validation automatically when a document is published. For more information, see "About Run-Time Validation for a Published Document" on page 573.

Performing XML Validation in Integration Server

Validate XML to ensure that the structure and content of the XML are valid. To validate an XML document, invoke the pub.schema:validate service. This service instructs the validation engine to validate an XML document by comparing it to a specified IS schema or an IS document type. The XML document is valid if it complies with the structural and content constraints described in the IS schema or IS document type. The pub.schema:validate service returns a string that indicates whether validation was successful and an *errors* variable (an IData array) that contains any validation errors.

By validating an XML document, you can ensure that the XML document contains the elements and attributes that:

- Are organized in the structure or format that subsequent services expect
- Contain values that are of the data type that subsequent services expect

For example, if you receive an XML document containing new employee information, you might want to validate the information to ensure it contains an employee name, address, telephone number, and date of birth before executing subsequent services. Additionally, you might want to validate the date of birth to make sure that it conforms to a specific date format.

Integration Server performs XML validation for web service requests that it receives an for web service requests that it produces automatically. Integration Server validates the request and response against the collection of IS elements that represent the WSDL on Integration Server, such as IS document types, IS schemas, service signatures, and the web service descriptor.

Note:
When validating supplied XML, if the XML contains an element defined to be of simple type with a pattern constraining facet, by default, Integration Server uses Perl pattern matching to evaluate element content. However, if watt.core.datatype.usejavaregex is set to true, during XML validation, Integration Server uses the Java regular expression compiler and Integration Server performs pattern matching as described by java.util.regex.pattern.

Note:

If you want Integration Server to skip validation of references to elements that are not namespace qualified made from within a namespace- qualified element, set the `watt.core.validation.skipNoNamespaceReference` server configuration property to `true`.

Performing Validation from within a Java Service

You can use built-in services pub.schema:validate and pub.schema:validatePipeline to perform validation from within a Java service. In the following example, the pub.schema:validate service is used to validate the results of the pub.xml:xmlNodeToDocument service against a specification for an OAG purchase order.

```
    .
    .
    .

    IData validInput;
    IData dtrResult;
    .
    .
    .

// initialize the folder and document type name to point to a document
// type that exists on the server
    String ifc = "OAG.PO"
    String rec = "purchaseOrder"

// put the result from the xmlNodeToDocument service (i.e, the object to
// be validated) into the key named <object>
    IDataCursor validCursor = validInput.getCursor();
    IDataCursor dtrCursor = drtResult.getCursor();

if (dtrCursor.first("boundNode")) {
    // assumption here that there's data at the current cursor position
    validCursor.insertAfter( "object", dtrCursor.getValue() );
}

dtrCursor.destroy();

// set <conformsTo> parameter to point to the document type to validate
// against this document type must exist on the server.
validCursor.insertAfter( "conformsTo", ifc+":"+rec );


// set the <maxErrors> parameter to the number of allowed validation
// errors
validCursor.insertAfter( "maxErrors", "1000" );
validCursor.destroy();

// invoke pub.schema.validate to validate contents of <object>
IData validResult = context.invoke("pub.schema", "validate", validInput);

// check <isValid> to see whether <object> is valid and process
// accordingly
IDataCursor validCursor = validResult.getCursor();
if(validCursor.first("isValid"))
{
    if (IDataUtil.getString(validCursor).equals("false"))
```

```
    {
        IData [] vr = IDataUtil.getIDataArray(validCursor, "errors");
        System.out.println ( vr.length+" ERROR(s) found with example");
        for (int j=0; j < vr.length; j++ )
        {
            System.out.println( vr[j].toString() );
        }
    }
}
validCursor.destroy();
      .      .      .
```

For additional information about pub.schema:validate and pub.schema:validatePipeline, see the Schema services in the *webMethods Integration Server Built-In Services Reference*.

Validation Errors

During data validation, the validation engine generates errors when it encounters values that do not conform to the structural and content constraints specified in the blueprint. The format in which the validation engine returns errors depends on whether validation was performed using the built-in services or by checking the declared input and output parameters for the service.

- When the validation engine performs data validation by executing the built-in services pub.schema:validate or pub.schema:validatePipeline, errors are returned in the errors output variable (an IData list). For each validation error, the *errors* variable lists the error code, the error message, and the location of the error.
- When the validation engine performs validation by comparing run-time data to the declared input and output parameters, the validation engine returns all the validation errors in a string. This string contains the error code, error message, and error location for each error found during input/output validation.

Validation Exceptions

If you use the pub.schema:validate and pub.schema:validatePipeline services to perform data validation, you can determine whether the service should succeed or fail if the data being validated is invalid. You might want a service to succeed even if the data is invalid. In the pub.schema:validate and pub.schema:validatePipeline services, the value of the *failIfInvalid* input variable determines whether a service fails because of an invalid object.

If the pub.schema:validate and pub.schema:validatePipeline service fails, Integration Server throws a validation exception. A validation exception is generated if one of the following is true:

- Errors are detected in the object (XML node, pipeline, or document (IData object)) that is passed (for example, null value).
- The basic validation contract is violated (for example, a binary tree is passed instead of a document (IData object) as expected).
- You specify that the service should fail if the object to be validated (XML node, pipeline, or document (IData object)) did not conform to the IS schema or IS document type (for example,

failIfInvalid = true). If this is the reason for the exception, Integration Server inserts the validation errors into the exception message.

Preventing Running Out of Memory Error During Validation

During validation of an XML node, a large maxOccurs value for an element in the IS schema used as the blueprint can cause an out of memory error or a stack overflow. To prevent a stack overflow or out of memory error, you can set a threshold value for maxOccurs. When the validation engine encounters a maxOccurs value greater than the threshold value, it proceeds as if the maxOccurs value was equal to 'unbounded'.

To set a maxOccurs threshold value, you can edit the server configuration parameter watt.core.schema.maxOccursThresholdValue. By default, this parameter does not have a value.

To set a maxOccurs threshold value

1. Start Integration Server and open the Integration Server Administrator.
2. In the **Settings** menu of the navigation area, click **Extended**.
3. Click **Edit Extended Settings**. The server displays the **Extended Settings** screen.
4. In the text area under **Extended Settings**, type `watt.core.schema.maxOccursThresholdValue=`*`value`* where *`value`* is the number you want to use as the maxOccurs threshold.
5. Click **Save Changes**.

14 Building Java Services

- Overview of Building Java Services 304
- Java Service Editor 305
- Service Development Projects in the Local Workspace 308
- How Java Services Are Organized on Integration Server 309
- Creating a Java Service 310
- Using an IData Object for the Java Service Input and Output 312
- Generating Java Code from Service Input and Output Parameters 314
- Editing an Existing Java Service 316
- Adding Classes to the Service Development Project 317
- Compiling a Java Service 319
- Generating Code a Java Service Can Use to Invoke a Specified Service 320
- Deleting a Java Service 322

This topic describes the use of Java services in a Service Development Project and how to use the Java Service Editor to create and edit Java services.

Overview of Building Java Services

Before you can create Java services using the Java Service Editor, you must meet the following prerequisites:

- Integration Server must have a Java compiler that is the same version as the Java compiler used in the Designer local workspace.
- Designer must have a connection to the Integration Server on which you want the Java service to reside.

In Designer you use the Java Service Editor to build Java services. For more information, see "Java Service Editor" on page 305. The following are the basic tasks you perform to create a Java service:

Task 1 Ensure that the IS package and folder in which you want to create the Java service exists.

If not, create them. For more information, see "Package and Folder Requirements" on page 159.

Task 2 Use Designer to add the Java service element. For more information, see "Creating a Java Service" on page 310.

Designer creates a Service Development Project in your local workspace for the Java service. For more information, see "Service Development Projects in the Local Workspace" on page 308.

Do the following to build the logic for the Java service:

- Define the input and output parameters for the service. For more information, see "About the Service Signature" on page 159.
- Optionally, generate starter code for the service based on the declared input and output parameters. For more information, see "Generating Java Code from Service Input and Output Parameters" on page 314.
- Add additional Java code and modify the generated Java code as necessary. You can use the webMethods Integration Server Java API in your service. For more information, see the *webMethods Integration Server Java API Reference*.

Task 3 Provide classes required to compile the Java service. You add any additional third-party classes to:

- Service Development Project in Designer so that Designer can locally compile the service. For more information, see "Adding Classes to the Service Development Project" on page 317.

- Integration Server so that the server can compile the service. For more information, see information about managing IS packages and how Integration Server stores IS package information in *webMethods Integration Server Administrator's Guide*.

Task 4 Compile the Java service. Designer automatically compiles the service when you save it. For more information, see "Compiling a Java Service" on page 319.

Task 5 Debug the Java service. For more information, see "Debugging Java Services" on page 455.

Designer also provides the ability for you to generate code that invokes a Java service. You can generate code that a client would use to invoke the Java service and code that another service would use to invoke the Java service. For more information, see "Building a Java Client" on page 910 and "Generating Code a Java Service Can Use to Invoke a Specified Service" on page 320.

Java Service Editor

Use the Designer Java service editor to create new Java services and to edit existing Java services. If you attempt to edit a Java service that you do not have locked for edit, you can still open it in the Java service editor. However, the source code, properties, inputs, and outputs will be read only.

The Java service editor has four tabs:

- **Source** tab contains the code for the Java service. For more information about the Source tab, see "Source Tab" on page 305
- **Input/Output** tab contains the input and output signature of the Java service. For more information about declaring the input and output parameters for a service, see "About the Service Signature" on page 159.
- **Logged Fields** tab indicates the input and output parameters for which Integration Server logs data. For more information about logging the contents of input and output fields, see "Logging Input and Output Fields" on page 185.
- **Comments** tab contains the comments or notes, if any, for the Java service.

Note:
You can use the Designer Java service editor to edit Java services that you created in Developer. Additionally, you can use Designer to edit Java services you created with your own IDE, provided that you properly commented them as described in "Building Java Services in Your Own IDE" on page 323 and "Adding Comments to Your Java Code for the jcode Utility" on page 327.

Source Tab

You specify the code for the Java service on the **Source** tab, which extends the standard Eclipse Java editor. Because the Eclipse Java editor requires source files to be in the local workspace, Designer also requires source files to be in the local workspace. To achieve this, Designer adds Java classes to a Service Development Project, which is a project with extensions to support Java

services. For more information, see "Service Development Projects in the Local Workspace" on page 308.

The full capabilities of the Eclipse Java editor are available, for example, source formatting, code completion, etc. However, unlike the Eclipse Java editor, the Designer Java service editor protects the sections of a Java service that contain required code to prevent structural damage to the service. The following illustrates the contents of the **Source** tab for a newly created service.

Java package definition

```
package orders.orderStatus
```

Add additional imports here

```
import com.wm.data.*;
import com.wm.util.Values;
import com.wm.app.b2b.server.Service;
import
com.wm.app.b2b.server.ServiceException;
```

Class definition

```
public final class orderStatus_checkStatus_SVC
```

Add extends and implements here

Primary method definition

```
{
     /**
      * The primary method for the java service
      *
      * @param pipeline
      *            The IData pipeline
      * @throws ServiceException
      */
     public static final void
            orderStatus_checkStatus(IData pipeline)
            throws ServiceException {
```

Add source code for the primary Java service method here

```
     }

     // --- <<IS-BEGIN-SHARED-SOURCE-AREA>> ---
```

Add shared code here

```
       // --- <<IS-END-SHARED-SOURCE-AREA>> ---
     }
```

Final "}"

```
}
```

Protected Sections of a Java Service

The Java service editor shades the protected sections of the Java service. By default, it uses gray for the shading; you can update your preferences to select a different color. For more information, see "Java/C Service Editors Preferences" on page 945. The sections of the Java service that the Java service editor protects are:

- **Java package definition**, which is the required code that defines the Java package for the Java service.
- **Class definition**, which is the required code that defines the final class for the Java service.
- **Primary method definition**, which is the required code that defines the static and final method for the Java service. It defines a single input parameter, an IData object.

  The IData object is the universal container that services use to receive input from and deliver output to other programs. It contains an ordered collection of key/value pairs on which a service operates. An IData object can contain any number of key/values pairs.

  You define the data to pass into the service via the IData object by defining input parameters on the **Input/Output** tab of the editor. You add code to the primary method that modifies the key/value pairs in the IData object. The IData object then becomes the output of the service. The service returns the output parameters you define on the **Input/Output** tab.

  Note:
  You can set the Java service editor preferences so that Designer uses `Values in` and `return out` for the input/output rather than an IData object. For more information, see "Java/C Service Editors Preferences" on page 945.

- **Final brace "}".** The Java service editor does not allow you to add code after the final brace "}".

Editable Sections of a Java Service

You can modify the following sections of the Java service:

- **imports**, where you can specify the names of additional Java packages whose classes you want to make available to the current class.

  Note:
  By default, Designer adds some required imports that you cannot delete. Although the Java service editor will allow you to remove the imports, when you save the service, Designer adds the required imports back to the service.

- **extends**, where you can specify a super class for the implementation.
- **implements**, where you can specify the Java interfaces that the Java service implements.
- **source code**, where you add the code for the primary Java service method.
- **shared code**, where you can specify declarations, methods, etc. that will be shared by all services in the current folder.

Note:
You cannot enter or paste special characters including '{' in the extends or implements section of a Java service.

The toolbar and icons that the **Source** tab uses are the same as the buttons and icons used in the standard Eclipse Java editor. For a description, see the *Eclipse Java Development User Guide*.

Service Development Projects in the Local Workspace

Because the Java service editor depends on files residing in the local workspace, Designer creates a Service Development Project in the local workspace to store files associated with a Java service. Also, if a Java service requires additional class or jar files so that Designer can compile the service, you add class and jar files to the Service Development Project.

When you first open a Java service, either when you edit it for the first time or by initially creating it, Designer adds a Java class associated with the Java service to a Service Development Project. If a Service Development Project does not already exist for a Java service, Designer creates one. You can view Service Development Projects using the Project Explorer view, the Package Explorer view, or the Navigator view.

About the Service Development Project Name

The Service Development Projects correspond to IS packages. To ensure that the project names are unique, Designer uses the following naming convention, where *packageName* is the name of the IS package where the service resides, *hostName* is the host name of the Integration Server on which the service resides, and *portNumber* is the port number of the Integration Server:

<*packageName*>[<*hostName*>_<*portNumber*>]

For example, if the host name of the Integration Server is “ServerA”, its port is “5555” and the IS package is named “MyPackage”, the Service Development Project will have the following name:

MyPackage[ServerA_5555]

If Designer has a second Integration Server connection to a server with host name “ServerB” that also uses port “5555” and has a Java service in an identically named package, “MyPackage”, the Service Development Project name will be unique in the workspace because it includes the server identification. In this case, the Service Development Project will have the following name:

MyPackage[ServerB_5555]

Format of a Service Development Project

The Service Development Project contains:

- JRE system library
- “src” folder that contains Java packages

  Designer creates Java packages that correspond to a Java service's IS namespace and places them within the “src” folder of the project. For example, if a Java service resides in the folder folderA.folderB, Designer creates the Java package “folderA.folderB” within the Service Development Project.
- Default .jar files that Designer includes in the project’s classpath

  Designer adds several default .jar files to the project. These files, for example, IS_CLIENT and IS_SERVER, are listed in uppercase.

- "class" folder where you can add any additional Java classes that your Java services require; see "Adding Classes to the Service Development Project" on page 317
- "lib" folder where you can add any additional Java classes that are packaged in jar files; see "Adding Classes to the Service Development Project" on page 317

Note:
You might still need to add additional classes and jar files to Integration Server so that Integration Server can compile the service. For more information, see information about managing IS packages and how Integration Server stores IS package information in *webMethods Integration Server Administrator's Guide*.

The following shows the format of a Service Development Project and an example.

Format

```
- projectName
   + JRE System Library
   - src
      +javaPackageName(1)
      .
      .
      .
      +javaPackageName(n)
   + classes
   + defaultJarFile(1)
   .
   .
   .
   + defaultJarFile(n)
   + lib
```

Example

```
- MyPackage[ServerA_5555]
   - JRE System Library
   - src
      - folderA
      - folderA.folderB
   + classes
   + IS_CLIENT
   + IS_SERVER
   .
   .
   .
   + lib
```

How Java Services Are Organized on Integration Server

A Java service is a public static method in a Java class file on Integration Server. Java services follow a simple naming scheme:

- The service name represents the Java method name.
- The interface name represents the fully-qualified Java class name.

Because Java class names cannot contain the "." character, services that reside in nested folders are implemented in a class that is scoped within a Java package. For example, a service named recording.accounts:createAccount is made up of a Java method called createAccount in a Java class called accounts within the recording package.

All Java services that reside in the same folder are methods of the same class.

When you build a Java service with Designer, the system automatically combines your service into the class file associated with the folder in which you created it.

Creating a Java Service

Use the Designer Java service editor to create a new Java service. Before you can create a Java service, make sure the following items are true:

- Designer is using the Service Development perspective. If not, switch to it by selecting **Window > Open Perspective > Service Development**.
- The Integration Server on which you want the Java service to reside is running and that it is connected to Designer.
- The IS package and folder in which you want to create the Java service already exists. For more information, see "Package and Folder Requirements" on page 159.
- All Java services in the folder in which you want to create the new service are unlocked. Alternatively, you can ensure that you have all the Java services locked. For more information, see "Guidelines for Locking Java and C/C++ Services" on page 90.
- If you want to use Unicode characters in the Java service, change the **Text file encoding** preference. To do so, select **Window > Preferences > General > Workspace** and for **Text file encoding** clear **Default (Cp1252)**, select **Other**, and then select or type a new encoding.

To create a Java service

1. In Designer: **File > New > Java Service**.
2. In the Create New Java Service wizard, expand the IS package in which you want the service to reside and select the folder in which you want to create the service.
3. In the **Element name** field, type the name you want to assign to the Java service.
4. If you have a template you want to use to initialize a default set of properties for the service, select if from the **Choose template** list.
5. Click **Finish**.
6. Specify the input parameters and output parameters for the Java service on the **Input/Output** tab. For more information, see "About the Service Signature" on page 159.
7. Optionally, specify usage notes or comments in the **Comments** tab.
8. Specify service properties using the Properties view. For more information, see:
   - "About Service Run-Time Parameters" on page 165
   - "About Automatic Service Retry" on page 178
   - "About Service Auditing" on page 180

- "About Universal Names for Services or Document Types" on page 191
- "About Service Output Templates" on page 195.

9. Optionally, generate starter code for the service based on the declared input and output parameters. For more information, see "Generating Java Code from Service Input and Output Parameters" on page 314.

10. Add and modify the Java code on the **Source** tab.

    You can use the webMethods Integration Server Java API in your service. For more information, see *webMethods Integration Server Java API Reference*.

11. Select **File > Save**.

    Designer compiles the Java service on Integration Server and displays compilation error messages from the server in a popup window. Designer also writes the error messages to the Designer log file making them visible within the Error Log View.

    Designer also compiles the Java service locally in the Service Development Project. Additionally, if the workspace preference **Build Automatically** is selected, Designer rebuilds other classes in the Service Development Project at the same time. Designer adds compilation errors from the local compilation to the Problems view. If Problems view is not already open, you can open it by selecting **Window > Show View > Problems**. To view the line of code that caused the error, double click on the error in the Problems view and Designer shifts focus to the Java service editor, with the cursor positioned at the line of code that caused the error.

    Note:

    For more information, see "Compiling a Java Service" on page 319.

Notes about Creating and Editing Java Services in Designer

Keep the following points in mind when you use Designer to create or edit a Java service:

- When you create a new Java service, Designer adds a Java class associated with the Java service to a Service Development Project in your local workspace. If an appropriate Service Development Project that corresponds to the service's IS package does not yet exist, Designer creates one for the service. For more information, see "Service Development Projects in the Local Workspace" on page 308.
- Designer adds initial code to the Java service. For all Java services, Designer adds the Java package definition, class definition, primary method definition, and a minimum set of imports. If the service is the second or subsequent Java service created in the same IS folder, Designer also adds any shared code defined in other Java services in the IS folder, additional imports, extends, and implements.
- Because Designer is connected to Integration Server, when you save the service in Designer, your changes are also immediately saved in Integration Server.

- Additionally, when you save a Java service, Designer compiles it both in the Service Development Project in Designer and on Integration Server. When Designer compiles the service locally, by default, it also rebuilds other classes in the Service Development Project.
- If your Java service requires additional classes to compile, you must add them, either as individual class files or in jar files, to both the Service Development Project and to Integration Server. If you set up IS package dependencies for the Java service in Integration Server and there are classes and/or jar files in the IS packages required so that the service can compile, you must manually add them to Service Development Project. For more information, see "Adding Classes to the Service Development Project" on page 317. For more information about adding classes to Integration Server and how Integration Server stores package information, see *webMethods Integration Server Administrator's Guide*.
- When a folder contains multiple Java services, Designer adds an empty implementation of all of the Java services in the folder to each Java service. This allows a Java service in a folder to invoke another Java service in the same folder directly using *methodName*(pipeline) where *methodName* is the local name of the Java service.

Using an IData Object for the Java Service Input and Output

An IData object is the universal container that Java services use for service input and output. A Java service method signature takes exactly one argument of type IData, and the same IData object contains the output from the service. An IData object contains an ordered collection of key/value pairs on which a service operates. For a key/value pair:

- The key must be a String.
- The value can be any Java object (including an IData object).

Tip:
You can use Designer to generate code for getting input from and writing output to an IData object. After generating the code, you can copy and paste it into the Java service you are creating. For more information, see "Generating Java Code from Service Input and Output Parameters" on page 314.

Sample Code for Getting and Setting Input Values

A Java service is a method that is public and static. It takes a single instance of com.wm.data.IData as input and must place output in the same IData object. The following code shows a sample:

```
public final static void myservice (IData pipeline)
     throws ServiceException
           {
     return;
     }
```

When the Java service is invoked, Integration Server passes the IData object to it. The service needs to get the input values it needs from the key/value pairs within the IData object. The following sample code uses methods of the IDataCursor class to position the cursor and uses the getValue method to get the input value of the *myVariable* input variable from the IData object:

```
public final static void myservice (IData pipeline)
```

```
     throws ServiceException
{
   IDataCursor myCursor = pipeline.getCursor();
   if (myCursor.first( "inputValue1" )) {
       String myVariable = (String) myCursor.getValue();
        .
        .
   }
   myCursor.destroy();
         .
         .
   return;
   }
```

A service returns output by inserting it into the same IData object that was used for the input values. All of the service outputs must be written to the IData object. For example:

```
public final static void myservice (IData pipeline)
     throws ServiceException
  {
   IDataCursor myCursor = pipeline.getCursor();
   if (myCursor.first( "inputValue1" )) {
       String myVariable = (String) myCursor.getValue();
             .
             .
   }
   myCursor.last();
   myCursor.insertAfter( "outputValue1", myOutputVariable );
   myCursor.destroy();
   return;
  }
```

Note: Integration Server passes everything that the Java service puts into the pipeline (that is, the IData object) as output, regardless of what is declared as its input/output parameters. Declaring a service's input and output parameters does not filter what variables the service actually receives or outputs at run time. It simply provides a formal description of what the service requires as input and produces as output.

Creating IData Objects

Use the IDataFactory class to create IData objects. For example:

```
public final static void myservice (IData pipeline)
     throws ServiceException
           {
            myIDataObject = IDataFactory.create();
            IDataCursor myCursor = myObject.getCursor();
            myCursor.insertAfter("VA", new Double("0.045"));
            myCursor.insertAfter("MD", new Double("0.05"));
            myCursor.insertAfter("DE", new Double("0.0"));
     return;
     }
```

Getting and Setting Elements in an IData Object

Use the IDataCursor class to get values from and set values into IData elements (that is, key/value pairs). Getting or setting values in IData elements takes two steps.

First, position the cursor at the IData element you want to get or set. The IDataCursor class contains methods for performing basic cursor operations such as placing the cursor at the first, last, or next element in an IData object.

After positioning the cursor, use the getValue or setValue methods to read or write the value of the element, respectively. This class also provides methods for inserting new elements, getting key names, and deleting elements.

For more information about using the cursor classes, see *webMethods Integration Server Java API Reference*.

Generating Java Code from Service Input and Output Parameters

If you know the set of input and output parameters that a Java service will use before you start coding it, you can declare the service's input/output parameters first and generate Java code from it. The generated code obtains the specified input values from the service's input signature and assigns them to variables in your service. It also puts the set of output values into the output signature.

You do not have to generate code for all the input and output parameters. You can choose to generate code for only the input parameters, only the output parameters, or you can select one or more input/output parameters for which to generate code.

When Designer generates code from the service input/output parameters, it puts the code on the clipboard. From there, you can paste it into the **Source** tab of your service, or if you are building a Java service in your own IDE, you can paste it into the service in your IDE. After pasting the generated code, you can modify it as necessary.

To generate starter code from a Java service's input/output parameters

1. Open the Java service by double clicking it in the Package Navigator view.

2. Select the **Input/Output** tab and define the inputs and outputs for the service if they are not already specified. For more information, see "About the Service Signature" on page 159.

3. If you want to generate code for a subset of the input/output parameters, on the **Input/Output** tab select the parameters for which you want to generate code. To select more than one variable, press the CTRL key as you select parameters.

4. Right click in the editor to view the context menu, and select **Generate Code**.

5. In the Code Generation window, select **For implementing this service** and click **Next**.

6. For **Specification**, select the **Input** and/or **Output** check boxes to reflect the parameters for which you want to generate code.

7. For **Which Fields?** select one of the following:

   - **All Fields** if you want to generate code for all of the parameters identified by your **Specification** selection.
   - **Selected Fields** if you want to generate code for only the parameters you selected before starting the code generation.

8. Click **Finish**. Designer generates code and places it on the clipboard.

9. Select the **Source** tab. Alternatively, if you are building a Java service using your own IDE, open the Java service in your IDE.

10. Paste the contents of the clipboard into your source code.

Example of Java Code Generated from Service Signature

Suppose you have a service that has the following input and output parameters:

Input Parameters		Output Parameters	
String	input1	String	output1
Document	inDoc	Document	outDoc
String	in1	String	out1
	in2		out2
String		String	

The following shows code that Designer generated for the above input and output parameters:

```
// pipeline
IDataCursor pipelineCursor = pipeline.getCursor();
   String  input1 = IDataUtil.getString( pipelineCursor, "input1" );

   // inDoc
   IData inDoc = IDataUtil.getIData( pipelineCursor, "inDoc" );
   if ( inDoc != null)
   {
      IDataCursor inDocCursor = inDoc.getCursor();
       String  in1 = IDataUtil.getString( inDocCursor, "in1" );
       String   in2 = IDataUtil.getString( inDocCursor, "in2" );
      inDocCursor.destroy();
   }
pipelineCursor.destroy();

// pipeline
IDataCursor pipelineCursor_1 = pipeline.getCursor();
IDataUtil.put( pipelineCursor_1, "output1", "output1" );

// outputDoc
IData  outputDoc = IDataFactory.create();
```

```
IDataCursor outputDocCursor = outputDoc.getCursor();
IDataUtil.put( outputDocCursor, "out1", "out1" );
IDataUtil.put( outputDocCursor, "out2", "out2" );
outputDocCursor.destroy();
IDataUtil.put( pipelineCursor_1, "outputDoc", outputDoc );
pipelineCursor_1.destroy();
```

Editing an Existing Java Service

Use the Designer Java service editor to edit an existing Java service. You can use the Designer Java service editor to edit Java services that were created in Designer or Developer. Additionally, you can use Designer to edit Java services you created with your own IDE, provided that you properly commented them as described in "Building Java Services in Your Own IDE" on page 323 and "Adding Comments to Your Java Code for the jcode Utility" on page 327.

Before you can edit an existing Java service, make sure that:

- The Integration Server on which you the Java service resides is running and that it is connected to Designer.
- You have the Java service locked for edit. If you attempt to edit a Java service that you do not have locked for edit, you can still open it in the Java service editor. However, the source code, properties, inputs, and outputs will be read only.
- If you want to use Unicode characters in the Java service, change the **Text file encoding** preference. To do so, select **Window > Preferences > General > Workspace** and for **Text file encoding** clear **Default (Cp1252)**, select **Other**, and then select or type a new encoding.

To edit an existing Java service

1. Ensure the Designer is using the Service Development perspective. If not, switch to it by selecting **Window > Open Perspective > Service Development**.
2. Double click the Java service in the Package Navigator view to open it.
3. Update the input parameters and output parameters for the Java service on the **Input/Output** tab. For more information, see "About the Service Signature" on page 159.
4. Modify service properties using the Properties view. For more information, see:
   - "About Service Run-Time Parameters" on page 165
   - "About Automatic Service Retry" on page 178
   - "About Service Auditing" on page 180
   - "About Universal Names for Services or Document Types" on page 191
   - "About Service Output Templates" on page 195.
5. Modify the Java code on the **Source** tab.

You can use the webMethods Integration Server Java API in your service. For more information, see the *webMethods Integration Server Java API Reference*.

6. Select **File > Save**.

   Designer compiles the Java service on Integration Server and displays compilation error messages from the server in a pop up window. Designer also writes the error messages to the Designer log file making them visible within the Error Log View.

   Designer also compiles the Java service locally in the Service Development Project. Additionally, if the workspace preference **Build Automatically** is selected, Designer also rebuilds other classes in the Service Development Project at the same time. Designer adds compilation errors from the local compilation to the Problems view. You can open Problems view by clicking **Window > Show View > Problems**. To view the line of code that caused the error, double click on the error in the Problems view and Designer shifts focus to the Java service editor, with the cursor positioned at the line of code that caused the error.

   The first time you edit a Java service in your workspace, Designer adds a Java class associated with the Java service to a Service Development Project in your local workspace. If an appropriate Service Development Project that corresponds to the service's IS package does not yet exist, Designer creates one for the service. For more information, see "Service Development Projects in the Local Workspace" on page 308.

Adding Classes to the Service Development Project

If a Java service requires additional classes to compile, you must add them to the following locations:

- Service Development Project in the local workspace so that Designer can compile the service.
- Integration Server so that the server can compile the service. Designer does not automatically propagate classes that you add to the Service Development Project to Integration Server; you must add them to Integration Server manually. For more information about adding classes to Integration Server, see information about managing IS packages and how Integration Server stores IS package information in *webMethods Integration Server Administrator's Guide*.

Keep the following points in mind when adding classes to the Service Development Project:

- You add individual class files to the "classes" folder of the Service Development Project.
- If you have Java classes that are packaged together in jar files, you add the jar files to the "lib" folder of the Service Development Project.
- If you set up IS package dependencies for a Java service in Integration Server and there are classes and/or jar files in the IS packages required so that the service can compile, you must manually add them to Service Development Project.

To add classes and jar files to the Service Development Project

1. Open the Project Explorer view.

2. Expand the Service Development Project for the Java service.

   Service Development Project names use the following format, where *packageName* is the name of the IS package where the service resides, *hostName* is the host name of the Integration Server on which the service resides, and *portNumber* is the port number of the Integration Server:

   <*packageName*>[<*hostName*>_<*portNumber*>]

   For example, if you want to add class and jar files for the order.orderStatus:checkStatus service that resides in the IS package "Accounting" on the Integration Server with the host name and port number "ServerA:5555", you would expand the Service Development Project with the following name:

   Accounting[ServerA_5555]

   For more information, see "Service Development Projects in the Local Workspace" on page 308.

3. If you want to add class files to the Service Development Project, drag them from the file system into the "classes" folder of the Service Development Project in the Project Explorer view.

   When adding class files, ensure that you keep the structure of the Java package intact. For example, if you want to add com.accounting.orders.statusClass.class, you must first create the "com", "accounting", and "orders" folders within the "classes" folder as shown below:

   ```
   - classes
     - com
       - accounting
         - orders
   ```

   Then add the statusClass.class file to the "orders" folder.

   Important:
   The Java source files for these classes should not be maintained within the Service Development Project.

4. If you want to add jar files to the Service Development Project, drag them from the file system into the "lib" folder of the Service Development Project in the Project Explorer view.

   If you have the **Build Automatically** Workspace preference selected, after adding new class and/or jar files to the Service Development Project, Designer automatically rebuilds the project. If you have the **Build Automatically** preference turned off, you can force a rebuild by selecting **Project > Build Project**. You set the **Build Automatically** preference using **Window > Preferences > General > Workspace**.

   After the project is rebuilt, Designer removes the errors from the Problems view. However, the errors might still exist for the Folder class that resides in Integration Server. To correct the error, ensure Integration Server has access to the required class and jar files, open the Java service in the Designer, and save it again to force the compilation of the service on Integration Server.

Compiling a Java Service

When you save a Java service, Designer automatically compiles the Java service in the Service Development Project and on Integration Server.

Before a Java service can be compiled, keep the following requirements and points in mind:

- You must add any additional Java classes that the Java service requires to both the Service Development Project and to Integration Server. For more information, see "Adding Classes to the Service Development Project" on page 317. For more information about adding classes to Integration Server, see information about managing IS packages and how Integration Server stores IS package information in *webMethods Integration Server Administrator's Guide*.
- Integration Server must have a Java compiler that is the same version as the Java compiler used in the Designer local workspace.
- When compiling the Java service locally, Designer uses the default Java compiler settings. You can update these settings by updating the Service Development Project's Java Compiler properties.
- By default, the Service Development Project uses the default JRE that is configured for Designer. You can configure a different JRE using the Service Development Project's Java Build Path properties and setting a new JRE on the **Libraries** tab.

Important:
You do *not* need to use the jcode utility to compile and transfer the Java service to Integration Server. The jcode utility is only necessary when you are using an IDE other than Designer. For more information about building Java services using your own IDE, see "Building Java Services in Your Own IDE" on page 323.

To compile a Java service

1. If the service is not open in the Java service editor, open it by double clicking the Java service in the Package Navigator view.

2. Select **File > Save** to save and compile the Java service.

Designer displays compilation errors from compiling the service in:

- **Problems view** for compilation errors from locally compiling the service

  If Problems view is not already open, you can open it by selecting **Window > Show View > Problems**.

  To view the line of code that caused the error, double click on the error in the Problems view and Designer shifts focus to the Java service editor, with the cursor positioned at the line of code that caused the error.

- **Popup window** for compilation errors from Integration Server

Designer writes the error messages from the server to the Designer log file, making them visible within the Error Log View.

If you receive errors because the Java compiler cannot be found in Integration Server, ensure you have a Java compiler installed on the same machine as Integration Server and that you have added the location of the Java compiler to the system path.

Performance When Compiling a Java Service

When Designer compiles the service locally, by default, it also rebuilds other classes in the Service Development Project. If you notice slower performance when you save, you can prevent Designer from rebuilding the other classes by updating the workspace preferences. Note that sometimes it is only the first attempt to save a Java service that takes a long time and future compilations might go faster.

If you want to turn off the rebuild of other classes in the Service Development Project, select **Window > Preferences > General > Workspace** and clear the **Build Automatically** check box. This preference affects all projects in the workspace. If you turn off automatic builds, you can manually force a build by selecting **Project > Build Project**.

Generating Code a Java Service Can Use to Invoke a Specified Service

You can have Designer generate Java code that invokes a selected service, which you can then add to a Java service. Designer generates code that:

- Creates an IData object based on the service's declared input parameters
- Invokes the selected service passing it the IData object and catching exceptions
- Retrieves the returned IData object from the selected service
- Uses the key/value pairs in the returned IData object to assign variables based on the declared output parameters of the selected service

You do not have to generate code for all the input and output parameters. You can select to generate code for only the input parameters, only the output parameters, or you can select one or more input/output parameters for which to generate code.

When Designer generates code from the service input/output parameters, it puts the code on the clipboard. From there, you can paste it into a Java service and modify it as necessary.

Generating Java Code to Invoke a Service

To generate Java code to invoke a service

1. Open the service that you want to invoke by double clicking it in the Package Navigator view.

2. If you want to generate code for a subset of the input/output parameters, on the **Input/Output** tab select the parameters for which you want to generate code. To select more than one variable, press the CTRL key as you select parameters.

3. Right click in the editor to view the context menu, and select **Generate Code**.

4. In the Code Generation window, select **For calling this service from another service** and click **Next**.

5. For **Specification**, select the **Input** and/or **Output** check boxes to reflect the parameters for which you want to generate code.

6. For **Which Fields?** select one of the following:

   - **All Fields** if you want to generate code for all of the parameters identified by your **Specification** selection.
   - **Selected Fields** if you want to generate code for only the parameters you selected before starting the code generation.

7. Click **Finish**. Designer generates code and places it on the clipboard.

8. Paste the contents of the clipboard into a Java service.

Example of Java Code Generated for Invoking a Service

Suppose you have a service that has the following input and output parameters:

Input Parameters		Output Parameters	
String	input1	String	output1
Document	inDoc	Document	outDoc
String	in1	String	out1
String	in2	String	out2

The following shows code that Designer generated for the above input and output parameters:

```
// input
IData input = IDataFactory.create();
IDataCursor inputCursor = input.getCursor();
IDataUtil.put( inputCursor, "input1", "input1" );

// inDoc
IData inDoc = IDataFactory.create();
IDataCursor inDocCursor = inDoc.getCursor();
IDataUtil.put( inDocCursor, "in1", "in1" );
IDataUtil.put( inDocCursor, "in2", "in2" );
inDocCursor.destroy();
IDataUtil.put( inputCursor, "inDoc", inDoc );
inputCursor.destroy();
```

```
// output
IData  output = IDataFactory.create();
try{
   output = Service.doInvoke( "Folder2.subFolder", "selectedService",
            input );
}catch( Exception e){}
IDataCursor outputCursor = output.getCursor();
   String  output1 = IDataUtil.getString( outputCursor, "output1" );

   // outputDoc
   IData  outputDoc = IDataUtil.getIData( outputCursor, "outputDoc" );
   if ( outputDoc != null)
   {
      IDataCursor outputDocCursor = outputDoc.getCursor();
          String  out1 = IDataUtil.getString( outputDocCursor, "out1" );
          String  out2 = IDataUtil.getString( outputDocCursor, "out2" );
      outputDocCursor.destroy();
   }
outputCursor.destroy();
```

Deleting a Java Service

When deleting a Java service, keep the following points in mind:

- When you delete a Java service, Designer recompiles any Java services that remain in the source folder.
- When you delete a folder or the last Java service in a folder, Designer also deletes the shared source for that folder.
- When a package containing a Java service is deleted, Designer deletes the corresponding Java projects from the file system.

To delete a Java service

1. In Package Navigator view, select the Java service that you want to delete.
2. Select **Edit > Delete**.
3. If you have selected the **Confirm before deleting** check box in the Preferences dialog box for Package Navigator view, do the following:
   a. If the Java service you want to delete have unsaved changes, Designer prompts you to save changes. Select the Java service that you want to save and click **OK**.
   b. If there are IS asset dependencies for the Java service you are deleting, Designer indicates which items will be affected by the deletion. Click **Continue**.
4. Click **OK** to confirm the deletion.

15 Building Java Services in Your Own IDE

- How Java Services are Organized on Integration Server 324
- Requirements for the Java Service Source Code 326
- IData Object for Java Service Input and Output 326
- Adding Comments to Your Java Code for the jcode Utility 327
- Using the jcode Utility 331

As an alternative to creating Java Services using the Designer Java Service Editor, you can use your own IDE.

Note:
For information about creating Java services using Designer, see "Building Java Services" on page 303.

When you use your own IDE, you must create the Java code yourself, compile it, and store the compiled class file and other service information in Integration Server. To help you with these tasks, Integration Server provides the jcode utility.

The following describes the basic steps for building a Java service with your own IDE.

1. Understand how Java service are stored in Integration Server. For a description, see "How Java Services are Organized on Integration Server " on page 324.
2. Optionally create an empty Java service using Designer that you can use as a guideline for coding your own service. For more information, see "Building Java Services" on page 303.
3. Write the Java code for your service using your own IDE.
   - Define the input and output parameters for the service. The service must use an IData object for service input and output. For more information, see "IData Object for Java Service Input and Output" on page 326.
   - Ensure your code meets requirements described in "Requirements for the Java Service Source Code" on page 326.
   - Add comments to the code that identify various fragments, for example, imports or service inputs and outputs. For more information, see "Adding Comments to Your Java Code for the jcode Utility" on page 327. These comments are used by the jcode utility, which you use in the next step.
4. Use the jcode utility to compile the Java service and store its service information in Integration Server.
5. Reload the package to load the compiled Java service into memory so that it is executable.

How Java Services are Organized on Integration Server

A Java service is a public static method of a Java class file on Integration Server. Java services use the following naming scheme:

- The service name represents the Java method name.
- The Integration Server folder name represents the fully qualified Java class name.

All Java services that reside in the same Integration Server folder are methods of the same class.

Services that reside in nested folders are implemented in a class that is scoped within a Java package. The Java package name corresponds to the nested folder names that contain the Java service folder. For example, a service named recording.accounts:createAccount is made up of a Java method called createAccount in a Java class called accounts within the recording Java package.

When building a Java service in your own IDE, it is helpful to understand how Java services are stored in Integration Server. Integration Server stores information about services within its packages directory, specifically in the namespace (ns) and code directories of a package.

The Namespace Directory

Each package on Integration Server has a *namespace* directory, called "ns." The following is an example of the namespace directory for a package called "purch":

Integration Server_directory \instances*instance_name*\packages\purch\ns

The ns directory contains information about the services and folders in the package. For a service, the ns directory contains information about properties of the service (for example, statelessness) and the input and output parameters of the service (if they have been defined).

When you build a Java service in your own IDE, you must use the jcode utility to generate the service and folder information for the namespace directory, including the node.ndf file so that the Java service becomes a part of the Integration Server namespace. Additionally, Designer obtains the service information from the ns directory. After running the jcode utility to populate the namespace for a service, you can view and/or edit the service in Designer. For more information, see "Using the jcode Utility" on page 331 and "Using jcode frag/fragall to Split Java Source for Designer " on page 335.

Important:
Although you might want to examine the contents of the Integration Server namespace directories, do not manually modify this information. Only modify this information using the appropriate Software AG tools and/or utilities. Inappropriate changes, especially to the ns directory of the WmRoot package, can disable Integration Server.

The Code Directory

The source and compiled code for a Java service is stored in the code directory of a package.

- Each package on the server has a code\source subdirectory that holds the Java source code for that package, if it is available. The following shows the path to the code\source subdirectory for the purch package:

  Integration Server_directory \instances*instance_name*\packages\purch\code\source

  When you build a Java service in your own IDE, save its source file in the code\source directory (subject to the normal Java constraints based on package declarations). You must name the files and intermediate directories according to the name of the service you are installing.

  The following shows the location of the source for the recording.accounts:createAccount service:

  Integration Server_directory
  \instances*instance_name*\packages\purch\code\source\recording\accounts.java

  The source code for the createAccount service is a method in accounts.java.

- The code\classes subdirectory contains the compiled code for a Java service (that is, the class file). The following shows the directory path to the classes in the purch package:

Integration Server_directory \instances*instance_name*\packages\purch\code\classes

When you build a Java service in your own IDE, you need to add the Java class file to the code\classes subdirectory. You can do so by using the jcode utility to compile the Java source. For more information, see "Using the jcode Utility" on page 331 and "Using jcode frag/fragall to Split Java Source for Designer " on page 335.

The following shows the location of the class file for the recording.accounts:createAccount service:

Integration Server_directory
\instances*instance_name*\packages\purch\code\classes\recording\accounts.class

The createAccount service is a method of the accounts class.

Requirements for the Java Service Source Code

When you build a Java service in your own IDE, ensure it meets the following requirements:

- The Java service must take a single IData object as input. For more information, see "IData Object for Java Service Input and Output" on page 326

  The following code shows the basic framework for a Java service that takes a single instance of com.wm.data.IData as input and returns output in the pipeline.

```
public final static void myservice(IData pipeline)
 throws ServiceException
     {
 return;
     }
```

  Note:
  Services can throw ServiceException. Do not call Service.throwError.

- The Java class must import the following Java packages.

  com.wm.data.*;

  com.wm.app.b2b.server.ServiceException;

  com.wm.app.b2b.server.Service;

- The Java class must be public.
- For performance reasons, it is recommended that you make the Java class final.

Note: Integration Server provides classes that you can use with Java services that you build. For a description of these classes, see *webMethods Integration Server Java API Reference*.

IData Object for Java Service Input and Output

An IData object is the universal container that Java services use for service input and output. A Java service method signature takes exactly one argument of type IData, and the same IData object

contains the output from the service. An IData object contains an ordered collection of key/value pairs on which a service operates. For a key/value pair:

- The key must be a String.
- The value can be any Java object (including an IData object).

When you build a Java service in your own IDE, you must code the service to get the input values from the IData object that gets passed into the method that implements the service. You must also code the service to write all output values to this IData object.

For more information about how to work with the IData object for input and output, including sample code for getting and setting values in the IData object, see "Using an IData Object for the Java Service Input and Output" on page 312.

Adding Comments to Your Java Code for the jcode Utility

When you build a Java service in your own IDE, you use the jcode utility to perform actions against the service. For more information, see "Using the jcode Utility" on page 331. Before you can use the jcode utility, you must add specially formatted Java comments (jcode tags) to the Java source code to designate the following segments of code:

- **Imports.** Add the following comments to mark the beginning and end of the import section.

```
// --- <<IS-START-IMPORTS>> ---
 imports
// --- <<IS -END-IMPORTS>> ---
```

- **Service definitions and service inputs and outputs.** Add the following comments to mark the beginning and end of the logic for one method in the class. This results in a Java service in Integration Server.

```
// --- <<IS-START(serviceName)>> ---
 service logic
// --- <<IS-END>> --
```

- **Shared code.** Add the following comments to mark the beginning and end of the shared code within the class.

```
// --- <<IS-START-SHARED>> --
 shared code
// --- <<IS-END-SHARED>> ---
```

For example, the following code fragment shows the tags used to mark the beginning and end of the import section.

```
    .

    .

    .

    // --- <<IS-START-IMPORTS>>
---
```

```
     import com.wm.data.*;
     import java.util.*;
     // --- <<IS-END-IMPORTS>>
---
     .
     .
     .
```

Template for Using jcode Tags in a Java Service

The following code is a template that shows the jcode tags (or comments) that you need to add to the Java source so that the jcode utility can identify code segments. For sample code that uses these jcode tags, see "Example of Code Commented for the jcode Utility" on page 329.

```
package Interface1;
/**
 * This is an example of an empty Java source code file that includes
 * the jcode tags (comments) that the jcode utility requires.
 */
import com.wm.app.b2b.server.Service;
import com.wm.app.b2b.server.ServiceException;
import com.wm.data.IData;
import com.wm.data.IDataCursor;
// --- <<IS-START-IMPORTS>>
---
// --- <<IS-END-IMPORTS>>
---
public class Interface0
{
     public static void Service1 (IData pipeline)
     throws ServiceException
     {
          // --- <<IS-START(Service1)>>
---
          // --- <<IS-END>> --
```

```
          return;
     }
     public static void Service2 (IData pipeline)
     throws ServiceException
     {
          // --- <<IS-START(Service2)>>
---
          // --- <<IS-END>> ---
          return;
     }
          // --- <<IS-START-SHARED>>
--
          // --- <<IS-END-SHARED>>
---
}
```

Example of Code Commented for the jcode Utility

The following is a complete example of Java source code that includes jcode tags (comments) that the jcode utility requires.

```
package recording;
/**
* This is an example of Java source code that includes jcode tags
* (comments) that the Integration Serverjcode utility requires. Note that, unless
* noted otherwise, when using the frag or fragall command of the
* jcode utility, the utility strips out all comments in the Java source
*  file.
*/
/**
* == IMPORTS ==
* Wrap imports the START-IMPORTS and END-IMPORTS tags.
*/
// --- <<IS-START-IMPORTS>> ---
import com.wm.app.b2b.server.Service;
import com.wm.app.b2b.server.ServiceException;
import com.wm.data.IData;
import com.wm.data.IDataCursor;
import com.wm.data.IDataUtil;
import java.util.*;
// --- <<IS-END-IMPORTS>> ---
/**
* == CLASS NAMING ==
* This class contains the definition of all the Java services
* within the recording.accounts folder (note the declaration for the
* recording Java package at the top). Each service is defined by a
```

```
* method. The service will have the same name as the method name.
* /
public class accounts
{
    /* *
    * == INDIVIDUAL SERVICES ==
    * The createAccount service expects three parameters: a string
    * ("name", a string array ("references"), and a document. The
    * service returns two strings ("message" and "id") .
    *
    * Wrap the start and end of the service with the
    * IS-START(<serviceName>)and IS-END tags, where <serviceName>
    * is the method/service name.
    * Note that the two lines immediately before start tag and after
    * the end tags are mandatory.
    *
    * Also note the use of comments to establish a signature for the
    * service. Each signature line has the following format:
    *
    *  [direction] type:dimension:option name
    *
    *  where:
    *    - direction: is "i" for input or "o" for output
    *    - type:      is one of the following:
    *                 "field" for instances of java.lang.String
    *                 "document type" for instances of com.wm.data.IData
    *                 "object" for instances of any other class
    *    - option:    is one of the following:
    *                 "required" if the parameter is required
    *                 "optional" if the parameter is optional
    *    - name:      is the name of the parameter
    *
    * To indicate nesting, use a single "-" at the beginning of
    * each line for each level of nesting.
    * /
    public static void createAccount (IData pipeline)
       throws ServiceException
     {
       // --- <<IS-START(createAccount)>> ---
       // [i] field:0:required name
       // [i] field:1:required references
       // [i] record:0:required data
       // [i] - field:1:required address
       // [i] - field:1:required phone
       // [o] field:1:required message
       // [o] field:1:required id
       IDataCursor idc = pipeline.getCursor();
       String name = IDataUtil.getString(idc, "name");
       String [] refs = IDataUtil.getStringArray(idc, "references");
       IData data = IDataUtil.getIData(idc, "data");
       // Service logic that takes action on the input information
       // goes here. Note that when you use the jcode utility to
       // fragment the service, it does not strip comments inside
       // the service body. As a result, the comments are
       // preserved and will display if you use Designer
       // to view the service.
       idc.last();
       idc.insertAfter ("message", "createAccount not fully implemented");
       idc.insertAfter ("id", "00000000");
       idc.destroy();
```

```
      // --- <<IS-END>> -- -
      return ;
    }
   /* *
   * == COMPLEX SIGNATURES = =
   * The getAccount service takes a single string "id", and returns
   * a complex structure representing the account information.
   * Note the use of the helper functions (defined below).
   * /
   public static void getAccount (IData pipeline)
      throws ServiceException
    {
      // --- <<IS-START(getAccount)>> -- -
      // [i] field:0:required id
      // [o] record:1:required account
      // [o] - field:0:required name
      // [o] - field:1:required refs
      // [o] - record:0:required contact
      // [o] -- field:0:required address
      // [o] -- field:0:required phone
      IDataCursor idc = pipeline.getCursor();
      if(idc.first("id"))
       {
        try
         {
           String id = IDataUtil.getString(idc);
           IData data = getAccountInformation(id);
           idc.last();
           idc.insertAfter ("account", data);
         }
        catch (Exception e)
         {
           throw new ServiceException(e.toString());
         }
       }
      idc.destroy();
      // --- <<IS-END>> -- -
    }
   /* *
   * == SHARED SOURCE ==
   * Wrap the start and end of the shared code with the
   * IS-START-SHARED and IS-END-SHARED tags. The shared code includes
   * both global data structures and non-public functions that are
   * not exposed as services.
   */
   // --- <<IS-START-SHARED>> ---
   private static Vector accounts = new Vector( );
   private static IData getAccountInformation (String id)  {
     throw new RuntimeException ("this service is not implemented yet");
    }
   // --- <<IS-END-SHARED>> ---
}
```

Using the jcode Utility

When building a Java service in your own IDE, use the jcode utility to perform actions that Designer performs when using Designer to build a Java service. These actions include compiling the Java

service and saving it as code fragments that Designer uses to display the service in the Designer Java Service Editor.

Before you use the jcode utility, add specially formatted Java comments (jcode tags) to the Java source code to designate various segments of the source code. For more information, see "Adding Comments to Your Java Code for the jcode Utility" on page 327.

The following table describes the main functions of the jcode utility.

jcode Command	Use to...
make makeall	Examine a package to determine the source files that have been updated since the last compilation, then compile those source files and save the resulting class files in the classes directory of the package. ■ Use make to compile the source files in a single folder of a package. ■ Use makeall to compile the source files in all the folders of a package. For more information, see "Using jcode comp to Create Java Source from Fragments" on page 335.
frag fragall	Split the source files in a package into fragments that the jcode utility then stores in the namespace (ns) directory of the package. As a result, when you view the service in Designer, Designer displays the code from the updated fragments. ■ Use frag to fragment the source files in a single folder of a package. ■ Use fragall to fragment the source files in all the folders of a package. For more information, see "Using jcode comp to Create Java Source from Fragments" on page 335.
comp	Build a composite from the code fragments defined in the namespace (ns) directory of a package to rebuild the Java source. The jcode utility saves the resulting Java source is in the source directory of the package. **Note:** Use this command if the Java source is lost or corrupted in some way and you need to recreate it. For more information, see "Using jcode comp to Create Java Source from Fragments" on page 335.

When building a Java service in your own IDE, you use the two-step process of making (compiling) and fragmenting the source code often. To make these actions easier, the jcode utility supports the shortcut commands described in the following table. For more information about these shortcuts, see "Using jcode Shortcut Commands" on page 336.

jcode Shortcut	Use to...
update	Compile and fragment only source files that have changed for a single package.
upall	Compile and fragment only source files that have changed for all Integration Server packages.
hailmary	Compile and fragment all source files (whether they have changed or not) for all Integration Server packages.

Using jcode make/makeall to Compile Java Source

Use the jcode make or makeall command to examine source files for one or more folders in a package and compile the source files that have been modified since the last compilation. The jcode utility saves the resulting class files in the classes directory of the package. After compiling the Java source, reload the Integration Server package so that the next time a service is invoked, Integration Server executes the updated class file.

The jcode utility reports which files were compiled, as well as any errors that it encountered during the compiling process.

Important:
Before you can compile a Java service using the jcode utility, you must set the environment variable, IS_DIR, to point to the directory in which Integration Server is installed.

Specifying the Compiler the jcode make/makeall Command Uses

By default, the jcode utility uses one of the following to compile the Java source, based on the settings of the watt.server.compile and watt.server.compile.unicode server configuration parameters.

- If the watt.server.compile and watt.server.compile.unicode parameters are omitted from the Integration Server configuration or if they are empty, by default the jcode utility uses the JVM internal Java compile tool to compile the Java source.
- If the watt.server.compile parameter specifies a compiler command, the jcode utility uses that compiler command to compile Java source that is *not* stored in Unicode encoding.
- If the watt.server.compile.unicode parameter specifies a compiler command, the jcode utility uses that compiler command to compile Java source that is stored in Unicode encoding.

For more information about the watt.server.compile and watt.server.compile.unicode server configuration parameters, see *webMethods Integration Server Administrator's Guide*.

If you want the jcode utility to use a different compiler than the default, update the jcode.bat file, which resides in the following location:

Integration Server_directory \instances*instance_name*\bin

where *instance_name* is the name of the Integration Server instance.

In the jcode.bat file, update the line with the Java command to specify the watt.server.compile system property that is set to the compiler you want to use. The property must have the following format:

```
"-Dwatt.server.compile="path_to_your_java_compile -classpath {0}
-d {1} {2}"
```

For example:

```
"-Dwatt.server.compile="C:\java\jdk1.6.0_11\bin\javac -classpath {0}
-d {1} {2}"
```

Using this example, the Java command would be the following:

```
"%JAVA_DIR%\bin\java" -Dwatt.server.compile="C:\java\jdk1.6.0_11\bin\javac
-classpath {0} -d {1} {2}"
-classpath "%IS_DIR%\..\common\lib\ext\mail.jar;%IS_DIR%\..\common\lib\ext\
enttoolkit.jar;%IS_DIR%\..\common\lib\wm-g11nutils.jar;%IS_DIR%\..\common\
lib\ext\icu4j.jar;%IS_DIR%\..\common\lib\wm-isclient.jar;%IS_DIR%\lib\
wm-isserver.jar" com.wm.app.b2b.server.NodeUtil "%IS_DIR%" %1 %2 %3 %4 %5
```

The watt.server.compile property specifies the compiler command that you want Integration Server to use to compile Java services. For more information about this property, see the *webMethods Integration Server Administrator's Guide*.

Important:
If the Java source code contains any non-ASCII characters, set the property watt.server.java.source=Unicode | UnicodeBig | UnicodeLittle. The default value is file.encoding. When Unicode is set, the compile command line specified in the property watt.server.compile.unicode is used. The default value of this property is the following:

"javac -encoding Unicode -classpath {0} -d {1} {2}"

Command Syntax for jcode make

Use the jcode make command to compile Java source for a single folder of an Integration Server package (that is, a single Java source file).

```
jcode make package folder
```

- *package* is the name of an Integration Server package.
- *folder* is the name of the folder in the specified package. This folder contains the source you want to compile.

Command Syntax for jcode makeall

Use the jcode makeall command to compile Java source for all folders of an Integration Server package.

```
jcode makeall package
```

- *package* is the name of the Integration Server package containing the source code you want to compile.

Using jcode frag/fragall to Split Java Source for Designer

Use the jcode frag or fragall command to split Java source files for one or more folders in a package into fragments. The jcode utility saves the resulting fragments and service signature (input and output parameters) in the namespace (ns) directory of the package. Because Designer obtains the service information from the ns directory, after running the frag or fragall command, if you view the service in Designer, Designer displays the code from the updated fragments.

The jcode utility does not modify the original Java source.

Important:
Before you use the jcode utility to update the Java code fragments and service signature, you must add specially formatted Java comments (jcode tags) to the Java source code to designate various segments of the source code. For more information, see "Adding Comments to Your Java Code for the jcode Utility" on page 327.

Command Syntax for jcode frag

Use the jcode frag command to fragment the Java source for a single folder (that is, a single Java source file).

```
jcode frag package folder
```

- *package* is the name of an Integration Server package.
- *folder* is the name of the folder in the specified package. This folder contains the source you want to fragment.

Command Syntax for jcode fragall

Use the jcode fragall command to fragment the Java source for all folders of an Integration Server package.

```
jcode fragall package
```

- *package* is the name of the Integration Server package containing the source code you want to fragment.

Using jcode comp to Create Java Source from Fragments

Use the jcode comp command to build a composite from fragments in the namespace (ns) directory to rebuild a Java source file. The jcode utility saves the resulting Java source in the source directory of the package.

Important:
The existing source file, if there is one, is overwritten by the source file that the jcode utility produces. User locks in Designer will not prevent this because the jcode utility operates independently of locking functionality.

Note:
When building a Java service in your own IDE, you cannot use the comp command if you have not previously used the frag/fragall to split the source into fragments.

Command syntax for jcode comp

Use the jcode comp command to build a Java source file based on the current fragments for the Java service in the namespace (ns) directory of an Integration Server package.

```
jcode comp package folder
```

- *package* is the name of an Integration Server package.
- *folder* is the name of the folder in the specified package. This folder identifies the Java source that you want to rebuild.

Using jcode Shortcut Commands

When building a Java service in your own IDE, it is common to use the two-step process to make (compile) Java source, then fragment the Java source. As a result, the jcode utility supports the shortcut commands to make this process easier.

Command Syntax for jcode update

Use the jcode update shortcut command to compile and fragment only source files that have changed for a specified Integration Server package.

```
jcode update package
```

- *package* is the name of the Integration Server package containing the source code you want to compile and fragment.

Command Syntax for jcode upall

Use the jcode upall shortcut command to compile and fragment only source files that have changed for all Integration Server packages.

```
jcode upall
```

Command Syntax for jcode hailmary

Use the jcode hailmary shortcut command to compile and fragment all source files (whether they have changed or not) in all Integration Server packages.

```
jcode hailmary
```

16 Building Map Services

- What Is a Map Service? 338
- Building Map Services Using the Tree Tab or Graphical View Tab 338
- Creating a Map Service 340
- Setting Properties for a Map Service 341
- Working in the Graphical View tab 341
- Debugging Map Services 342

A *map service* is a service that is written in the webMethods flow language. A map service can be used to map document types of different formats.

A map service can be reused in different flow services. You can create a map service to perform a complex mapping of service signatures (input and output variable) and invoke this mapping service when the same transformation or mapping is required in other flow or Java services.

What Is a Map Service?

A map service, like a flow service, is a service that is written in the webMethods flow language that allows you to adjust the contents and structure of a pipeline.

As with a MAP step, you can use a map service to:

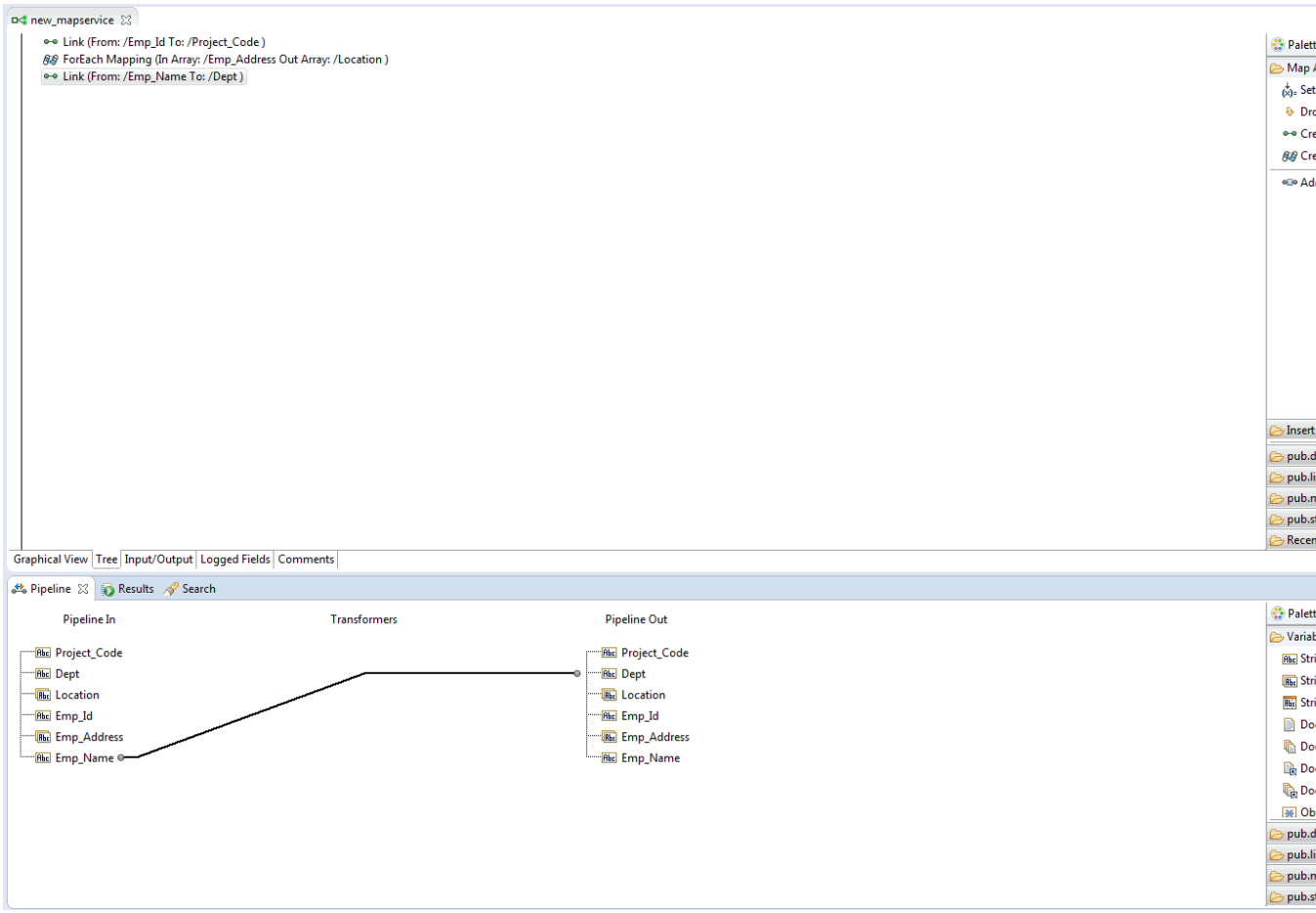
- Prepare the pipeline for use by a subsequent step in the map service by linking, adding, and dropping variables in the pipeline.
- Clean up the pipeline after a preceding step by removing fields that the step added but are not needed by subsequent steps.
- Move variables or assign values to variables in the pipeline.
- Initialize the input values for a map service.
- Invoke several services (transformers) in a single step.
- Map documents from one format to another. For example, you can map a document in an XML format to an ebXML format or a proprietary format.
- Create a ForEach mapping between the array variables in the pipeline.

Unlike a MAP step, a map service exists as an independent element in the Integration Server namespace, allowing the map service logic to be reused by other flow services. For example, if you regularly need to map from one format to another, you might create a map service that contains all of the logic to transform the data. Services that need the same data transformation, can invoke this mapping service. Similarly, if you have a set of complex mapping that needs to be performed repeatedly, then instead of placing the complex mapping in a MAP step or a series of MAP steps, you can create a single map service. You can invoke this map service when the other services require the same complex mapping.

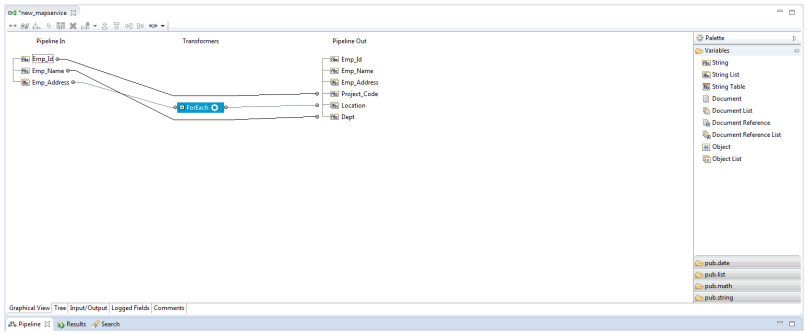
Building Map Services Using the Tree Tab or Graphical View Tab

In the map service editor, you can view and build map services using the Tree tab or Graphical view tab.

- On the Tree tab, Designer lists map actions from top to bottom. The Graphical Tree tab provides a more condensed view of a map service. When you click on a map action in the map service, the corresponding step is displayed in the Pipeline view.



- On the Graphical View tab, Designer provides a graphical representation of all the map actions involved in a map service.



Tree tab and Graphical View tab provide the same capabilities for building a map service hence, work in whichever tab you find easier to use. You can easily switch between the tabs when building a map service.

Creating a Map Service

Creating a Map Service

When you create an map service, you must perform the map actions manually.

To create a map service

1. In Designer: **File > New > Map Service**.
2. In the New Map Service dialog box, select the folder in which you want to save the map service.
3. In the **Element name** field, type a name for the map service using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
4. If you have a template you want to use to initialize a default set of properties for the service, select if from the **Choose template** list.
5. Click **Finish** to create the map service.
6. To perform a map action steps, perform the following in the map service editor:
   a. Perform one of the following:
      - Click the ▾ button next to ↵ on the map service editor toolbar and select the map action that you want to perform.
      - In the Palette view, select the map action that you want to perform and drag it to the map service editor.

      The corresponding map action dialog box appears.
   b. Perform one of the following map actions:

Map Action	Description
Set a Value	Assigns a value to the selected variable in the Pipeline In or Pipeline Out stage of map service. For details, see "Assigning a Value to a Pipeline Variable" on page 272. On the Set a Value dialog box , select the variables to which you want to assign a value and set the values.
Drop a Variable	Drops the selected variable from the Pipeline In or Pipeline Out stage of map service. For details, see "Dropping Variables from the Pipeline" on page 275.

Map Action	Description
	On the Drop a Variable dialog box, select the variable which you want to drop from the pipeline.
Create a Link	Creates a link between the variables defined in the map service. For details, see "Creating a Link Between Variables" on page 258. On the Create a Link dialog box, select the source and target variables between which you want to create a link.
Create a ForEach	Creates a ForEach mapping between array variables defined in the map service. For details, see "Creating a ForEach Mapping" on page 286. On the Create a ForEach Mapping dialog box, select the source and target array variables between which you want to create a ForEach mapping.
Add a Transformer	Inserts a service for use as a transformer in the map service. For details, see "Inserting a Transformer" on page 279. Service could be a built in service, previously built flow service, or a map service. On the Select... dialog box, select the required service to be added as a transformer in the map service.

7. Click **OK**.

   The updated map service appears on the Tree tab.

Setting Properties for a Map Service

Every map service is associated with a unique set of properties. The properties of a map service are displayed in the Properties view. You can specify the values of a map service in the Properties view.

Similarly, each map action has a set of properties. The properties of a map action are displayed in the Properties view. Values that you specify in the Properties view apply only to the selected map action in the editor.

Working in the Graphical View tab

Graphical Tree tab allows you to build or edit a map service. Graphical View tab provides a graphical representation of the entire map service. Changes made to any map action in the Tree tab reflect immediately in the Graphical Tree tab. Similarly, any changes made in the Graphical View tab reflect in the Tree tab.

In Graphical View tab, you can perform all the operations similar to the Pipeline View. For more information about the working with Pipeline View, see "Mapping Data in Flow Services" on page 251.

Debugging Map Services

Debugging in map service is similar to that in a flow service. Designer makes use of the same tools, commands, and processes to debug a map service as used in a flow service. For details on debugging a flow service, see "Debugging Flow Services" on page 429.

17 Building C/C++ Services

- The Java Code for a C/C++ Service 344
- Overview of Building C/C++ Services 344
- Prerequisites for Building C/C++ Services 345
- C/C++ Service Editor 346
- Service Development Projects in the Local Workspace 349
- How C/C++ Services Are Organized on Integration Server 351
- Creating a C/C++ Service 351
- Generating C/C++ Code from Service Input and Output Parameters 354
- Adding Classes to the Service Development Project 355
- Building the C/C++ Source Code 356
- Compiling the C/C++ Source Code 357
- Generating Code a C/C++ Service Can Use to Invoke a Specified Service 359
- Debugging C/C++ Services 360

A C/C++ service is a Java service that calls a C program that you have created. Designer generates the Java code needed to successfully call the C program.

You use Designer to build a set of starter files that you can use to create a C/C++ service.

These files include:

- A Java service that calls the C program that you have created.
- A C/C++ source-code template that you use to create your C program.
- A make file you use to compile the finished program and place it on the server.

The Java Code for a C/C++ Service

When you create a C/C++ service, Designer creates a Java service. This Java service calls the C program that you have created.

The C service is the means by which your C program is exposed to clients. The C service also supplies the input/output parameters for the C program, which makes it possible to include it in a flow service and link its inputs and output on the Pipeline view.

Designer generates all the Java code needed to successfully call your C program. You may add your own custom code to the C/C++ service editor if you want to execute any special procedures before or after executing the C program.

Overview of Building C/C++ Services

In Designer, you use the C/C++ service editor to build C/C++ services. For more information about the C/C++ service editor, see "C/C++ Service Editor" on page 346. The following are the basic tasks you perform to create a C/C++ service:

Task 1	Complete the prerequisite activities mentioned in "Prerequisites for Building C/C++ Services" on page 345.
Task 2	Use Designer to create the C/C++ service element. For more information, see "Creating a C/C++ Service" on page 351.
Task 3	Generate starter code for the service based on the declared input and output parameters. For more information, see "Generating C/C++ Code from Service Input and Output Parameters" on page 354.
Task 4	Add additional code or modify the generated code, if necessary. You can use the Integration Server C/C++API in your service. For more information, see *webMethods Integration Server C/C++ API Reference*.
Task 5	Specify service properties such as the run time settings, service retry, service auditing, and permissions using the Properties view. For more information, see "Building Services" on page 157.

Task 6	Provide classes required to compile the C/C++ service. You add any additional third-party classes to: ■ Service Development Project in Designer so that Designer can locally compile the service. For more information, see "Adding Classes to the Service Development Project" on page 355. ■ Integration Server so that the server can compile the service. For more information, see information about managing IS packages and how Integration Server stores IS package information in *webMethods Integration Server Administrator's Guide*.
Task 7	Compile the C/C++ service. Designer automatically compiles the service when you save it. For more information, see "Compiling the C/C++ Source Code" on page 357.
Task 8	Debug the C/C++ service. The primary way to debug a C/C++ service is to debug the Java class associated with the C/C++ service that Designer maintains in a Service Development Project. For more information, see "Debugging C/C++ Services" on page 360.

Prerequisites for Building C/C++ Services

Keep the following points in mind when creating C/C++ services using the C/C++ service editor:

- You must have a C/C++ compiler installed on the host where Integration Server is installed.
- You must complete the procedures specified in *Integration Server_directory* /sdk/c/README and/or *Integration Server_directory* /sdk/cpp/doc/README to build the platform support libraries needed by Integration Server and Designer.
- Designer must be connected to the Integration Server in which you want the C/C++ service to reside.
- The package in which you want to create the service must already exist. For more information about creating a package, "Creating a Package" on page 142. (If you do not have Developer or Administrator privileges, ask your server administrator to do this.)

  The directory for this package must contain a "code/libs" directory. When you compile your C/C++ service, the make file places the compiled service (a DLL) in this directory. If the package does not already have a code/libs directory, create one before you begin building the service.
- The folder in which you want to create the service must already exist. For more information, see "Creating New Elements" on page 41.
- The specification that you want to use to define the inputs and outputs for the service must exist. For more information about specifying a specification, see "Using a Specification as a Service Signature" on page 163.
- If you are running the Integration Server as an NT service, you must complete one of the following:

- Set the Windows system environment variable PATH to include *Integration Server_directory* \lib

  -OR-

- Copy the wmJNI.dll and wmJNIc.dll files located in *Integration Server_directory* \lib to the *Integration Server_directory*

C/C++ Service Editor

Use the Designer C/C++ service editor to create new C/C++ services and to edit existing C/C++ services.

The C/C++ service editor has four tabs:

- **Source** tab contains the code for the C/C++ service. For more information about the Source tab, see "Source Tab" on page 346
- **Input/Output** tab contains the input and output signature of the C/C++ service. For more information about declaring the input and output parameters for a service, see "About the Service Signature" on page 159.
- **Logged Fields** tab indicates the input and output parameters for which Integration Server logs data. For more information about logging the contents of input and output fields, see "Logging Input and Output Fields" on page 185.
- **Comments** tab contains the comments or notes, if any, for the C/C++ service.

Note:
You can use the Designer C/C++ service editor to edit the C/C++ services that you created in Developer. Additionally, you can use Designer to edit C/C++ services you created with your own IDE, provided that you properly commented them as described in "Building Java Services in Your Own IDE" on page 323 and "Adding Comments to Your Java Code for the jcode Utility" on page 327.

Source Tab

You specify the code for the C/C++ service in the **Source** tab, which extends the standard Eclipse Java editor. Because the Eclipse Java editor requires source files to be in the local workspace, Designer also requires source files to be in the local workspace. To achieve this, Designer adds Java classes to a Service Development Project, which is a project with extensions to support Java services. For more information, see "Service Development Projects in the Local Workspace" on page 349.

The full capabilities of the Eclipse Java editor are available. These include source code formatting and code completion. However, unlike the Eclipse Java editor, the Designer C/C++ service editor protects the sections of a C/C++ service that contain required code to prevent structural damage to the service. The following illustrates the contents of the **Source** tab for a newly created service.

Package definition

```
package orders.orderStatus;
```

Add additional imports here

```
import com.wm.data.*;
 import com.wm.util.Values;
 import com.wm.app.b2b.server.Service;
 import com.wm.app.b2b.server.ServiceException;
 import com.wm.app.b2b.server.Session;
 import com.wm.util.JournalLogger;
 import com.wm.util.DebugMsg;
```

Class definition

```
public final class checkStatus_SVC
```

Add extends and implements here

Primary method definition

```
{
    /**
      * The primary method for the C service
      *
      * @param in
       *
       The input Values      * @return The output Values
      */
     public static final Values checkStatus(Values in)
 {
           // --- <<IS-GENERATED-CODE-1-START>> ---
           Values out = in;
           // --- <<IS-GENERATED-CODE-1-END>> ---
           out = ccheckStatus(Service.getSession(),
                  in);
```

Add source code for the primary method here

```
// --- <<IS-GENERATED-CODE-2-START>> ---
           return out;
 // --- <<IS-GENERATED-CODE-2-END>> ---
   }
```

Add shared code here

```
// --- <<IS-BEGIN-SHARED-SOURCE-AREA>> ---0
    static {
      try {
       System.loadLibrary("orders_orderStatus");
       JournalLogger.log(DebugMsg.LOG_MSG,
             JournalLogger.FAC_PACKAGE,
             JournalLogger.INFO,
    "Loading
native library: orders_orderStatus");
         } catch (UnsatisfiedLinkError e)
 {
            JournalLogger.logError(DebugMsg.LOG_MSG,
            JournalLogger.FAC_PACKAGE,
                 e.getMessage());
        }
       }
       native static Values ccheckStatus(Session
 session, Values in);
 // --- <<IS-END-SHARED-SOURCE-AREA>> ---
```

Final "}"

```
}
```

Protected Sections of a C/C++ Service

The C/C++ service editor protects certain sections of the C/C++ service and marks these sections by highlighting them in a different color. By default, it uses gray for the shading but you can update your preferences to select a different color. For more information, see "Java/C Service Editors Preferences" on page 945. The sections of the C/C++ service that the C/C++ service editor protects are:

- **Package definition**, which is the required code that defines the Java package for the C/C++ service.
- **Class definition**, which is the required code that defines the final class for the C/C++ service.
- **Primary method definition**, which is the required code that defines the static and final method for the C/C++ service. It defines a single input parameter, a *Values* object.

  The Values object is the universal container that services use to receive input from and deliver output to other programs. It contains an ordered collection of key/value pairs on which a service operates. A Values object can contain any number of key/values pairs.

  You define the data to pass into the service via the Values object by defining input parameters on the **Input/Output** tab of the editor. You add code to the primary method that modifies the key/value pairs contained in the Values object. The Values object then becomes the output of the service. The service returns the output parameters you define on the **Input/Output** tab.
- **Final brace "}".** The C/C++ service editor does not allow you to add code after the final brace "}".

Editable Sections of a C/C++ Service

You can modify the following sections of the C/C++ service:

- **imports**, where you can specify the names of additional Java packages whose classes you want to make available to the current class.

  Note:
  By default, Designer adds some required imports that you cannot delete. Although you can remove the imports in the C/C++ service editor, Designer adds the required imports back when you save the service.

- **extends**, where you can specify a super class for the implementation.
- **implements**, where you can specify the Java interfaces that the C/C++ service implements.

  Note:
  You cannot enter or paste special characters including '{' in the extends or implements section of a C/C++ service.

- **source code**, where you add the code for the primary C/C++ service method.
- **shared code**, where you can specify declarations, methods, etc. that will be shared by all services in the current folder.

  Note:
  The shared code section of the C/C++ service editor contains the code that loads the library that contains the C/C++ program.

The toolbar and icons that the **Source** tab uses are the same as the buttons and icons used in the standard Eclipse Java editor. For a description, see the *Eclipse Java Development User Guide*.

Service Development Projects in the Local Workspace

Because the C/C++ service editor requires that some files exist in the local workspace, Designer creates a Service Development Project in the local workspace to store files associated with a C/C++ service. Also, if a C/C++ service requires additional class or jar files so that Designer can compile the service, you add class and jar files to the Service Development Project. Designer creates one Service Development Project per package containing a C/C++ service.

When you create a C/C++ service, Designer adds a Java class associated with the C/C++ service to a Service Development Project. If a Service Development Project does not already exist for a C/C++ service, Designer creates one. You can use the Project Explorer, Package Explorer, or Navigator views to view the Service Development Projects.

About the Service Development Project Name

The Service Development Projects correspond to Integration Server packages. To ensure that the project names are unique, Designer uses the following naming convention, where *packageName* is

the name of the IS package where the service resides, *hostName* is the host name of the Integration Server on which the service resides, and *portNumber* is the port number of the Integration Server:

<*packageName*>[<*hostName*>_<*portNumber*>]

For example, if the host name of the Integration Server is "ServerA", its port is "5555" and the IS package is named "MyPackage", the Service Development Project will have the following name:

MyPackage[ServerA_5555]

Format of a Service Development Project

The Service Development Project contains:

- JRE system library
- "src" folder that contains Java packages

  Designer creates Java packages that correspond to a C/C++ service's IS namespace and places them within the "src" folder of the project. For example, if a C/C++ service resides in the folder folderA.folderB, Designer creates the Java package "folderA.folderB" within the Service Development Project.
- Default .jar files that Designer includes in the project's classpath

  Designer adds several default .jar files to the project. These files are listed in uppercase (for example, IS_CLIENT and IS_SERVER).
- "class" folder where you can add any additional Java classes that your C/C++ service require; see "Adding Classes to the Service Development Project" on page 355
- "lib" folder where you can add any additional Java classes that are packaged in jar files; see "Adding Classes to the Service Development Project" on page 355

Note:
You might still need to add additional classes and jar files to Integration Server so that Integration Server can compile the service. For more information about managing IS packages and how Integration Server stores IS package information, see *webMethods Integration Server Administrator's Guide*.

The following shows the format of a Service Development Project and an example.

Format	Example
- *projectName* + JRE System Library - src +*javaPackageName(1)* . . . +*javaPackageName(n)* + classes + *defaultJarFile(1)* . . . + *defaultJarFile(n)* + lib	- MyPackage[ServerA_5555] + JRE System Library - src - folderA - folderA.folderB + classes + IS_CLIENT + IS_SERVER . . . + lib

How C/C++ Services Are Organized on Integration Server

A C/C++ service is a public static method in a Java class file on Integration Server. C/C++ services follow a simple naming scheme:

- The service name represents the Java method name.
- The interface name represents the fully-qualified Java class name.

Because Java class names cannot contain the "." character, services that reside in nested folders are implemented in a class that is scoped within a Java package. For example, a service named recording.accounts:createAccount is made up of a Java method called createAccount in a Java class called accounts within the recording package.

All C/C++ services that reside in the same folder are methods of the same class.

When you build a C/C++ service with Designer, the system automatically combines your service into the class file associated with the folder in which you created it.

Creating a C/C++ Service

Before you can create a C/C++ service, make sure the following conditions are true:

- The Integration Server on which you want the C/C++ service to reside is running and connected to Designer.
- You have locked all C/C++ services in the folder in which you want to create the new service. For more information, see "Guidelines for Locking Java and C/C++ Services" on page 90.
- If you want to use Unicode characters in the C/C++ service, you need to change the text file encoding preference. To do so, in the Workspace preferences, select **Other** under **Text file encoding** and select or type a new encoding.

To create a C/C++ service

1. In the Package Navigator view of Designer, select **File > New > C Service**.

2. In the New C Service dialog box, select the folder in which you want to save the service.

3. In the **Element name** field, type the name for the C/C++ service.

4. If you have a template you want to use to initialize a default set of properties for the service, select it from the **Choose template** list. If you want to apply the default properties to the service, select **Default** from the **Choose template** list. Click **Next**.

5. Select the platform that describes the machine on which your Integration Server is running (Designer needs to know this in order to build the right make file). Click **Next**.

6. Select the specification that defines the inputs and outputs for this service.

7. Click **Finish**. Designer refreshes the Package Navigator view and displays the new service in the C/C++ service editor.

8. Specify service properties using the Properties view.

To...	See...
Specify the service run-time parameters	"About Service Run-Time Parameters" on page 165
Configure the service to retry automatically if the service fails because of an ISRuntimeException	"About Automatic Service Retry" on page 178
Track when the service is started and completed and whether the service succeeded or failed	"About Service Auditing" on page 180
Assign a universal name to the service	"About Universal Names for Services or Document Types" on page 191
Assign an output template to the service	"About Service Output Templates" on page 195

9. Optionally, generate starter code for the service based on the declared input and output parameters.

   Designer adds initial code to the C/C++ service. For all C/C++ services, Designer adds the package definition, class definition, primary method definition, and a minimum set of imports. If the service is the second or subsequent C/C++ service created in the same IS folder, Designer also adds any shared code defined in other C/C++ services in the IS folder, additional imports, extends, and implements.For more information, see "Generating C/C++ Code from Service Input and Output Parameters" on page 354.

10. Add and modify the code on the **Source** tab. You can specify declarations, methods, etc. to the initial code that Designer generates.

    You can use the webMethods Integration Server Java API in your service. For more information, see *webMethods Integration Server Java API Reference*.

11. Optionally, specify usage notes or comments in the **Comments** tab.

12. Select **File > Save**.

    Designer compiles the C/C++ service on Integration Server and displays the compilation error messages from the server. Designer also writes the error messages to the Designer log file making them visible within the Error Log view.

    When you create a C/C++ service, Designer generates a source code file and a make file and places these files in the following directory:

    Integration Server_directory \instances*instance_name*\packages\packageName\code\source

    The names of the files will match the service name you specified in Designer. The source code file will be named <*servicename*>.c and the make file will be <*servicename*>.mak.

    Designer also compiles the C/C++ service locally in the Service Development Project. Additionally, if the workspace preference **Build Automatically** is selected, Designer rebuilds other classes in the Service Development Project at the same time. Designer adds compilation errors from the local compilation to the Problems view. If Problems view is not already open, you can open it by selecting **Window > Show View > Problems**. To view the line of code that caused the error, double click on the error in the Problems view and Designer shifts focus to the C/C++ service editor, with the cursor positioned at the line of code that caused the error. For more information, see "Compiling the C/C++ Source Code" on page 357.

    Note:
    If your C/C++ service requires additional classes to compile, you must add them, either as individual class files or in jar files, to both the Service Development Project and to Integration Server. If you have set up IS package dependencies for a C/C++ service and if the service requires classes or jar files in these IS packages to compile, you must manually add the classes or jar files to Service Development Project. For more information, see "Adding Classes to the Service Development Project" on page 355. For more information about adding classes to Integration Server and how Integration Server stores package information, see *webMethods Integration Server Administrator's Guide*.

Editing an Existing C/C++ Service

You can use the Designer C/C++ service editor to edit C/C++ services that were created in Designer or Developer. Additionally, you can use Designer to edit C/C++ services you created with your own IDE, provided that you properly commented them as described in "Building Java Services in Your Own IDE" on page 323 and "Adding Comments to Your Java Code for the jcode Utility" on page 327.

Keep the following points in mind when editing an existing C/C++ service:

- You have the C/C++ service locked for edit. If you attempt to edit a C/C++ service that you have not locked for edit, you can still open it in the C/C++ service editor. However, the source code, properties, inputs, and outputs will be read only.
- If you want to use Unicode characters in the C/C++ service, you need to change the text file encoding preference. To do so, in the Workspace preferences, select **Other** under **Text file encoding** and select or type a new encoding.

Important: Software AG recommends that you do not update the service signature of the C/C++ service in the Input/Output tab of the C/C++ service editor.

Generating C/C++ Code from Service Input and Output Parameters

If you know the set of input and output parameters that a C/C++ service will use before you start coding it, you can declare the service's input/output parameters first and generate C/C++ code from it. The generated code obtains the specified input values from the service's input signature and assigns them to variables in your service. It also puts the set of output values into the output signature.

You do not have to generate code for all the input and output parameters. You can choose to generate code for only the input parameters, only the output parameters, or you can select one or more input/output parameters for which to generate code.

When Designer generates code from the service input/output parameters, it puts the code on the clipboard. From there, you can paste it into your service and modify it as necessary.

Generating the code from the service input and output parameters is an optional task. It helps save time when you are reusing the input and output parameters of a C/C++ service.

To generate starter code from a C/C++ service's input/output parameters

1. In the Service Development perspective, open the C/C++ service by double clicking it in the Package Navigator view.
2. If you want to generate code for a subset of the input/output parameters, on the **Input/Output** tab, select the parameters for which you want to generate code. To select more than one variable, press the CTRL key as you select parameters.
3. Right click in the editor to view the context menu, and select **Generate Code**.
4. In the Code Generation dialog box, select **For implementing this service** and click **Next**.
5. For **Specification**, select the **Input** and/or **Output** check boxes to select the parameters for which you want to generate code.
6. For **Which fields?** select one of the following:

- **All fields** if you want to generate code for all of the parameters identified by your **Specification** selection.
- **Selected fields** if you want to generate code for only the parameters you selected before starting the code generation.

7. Click **Finish**. Designer generates code and places it on the clipboard.
8. Select the **Source** tab.
9. Paste the contents of the clipboard into your source code.
10. Save the C/C++ service.

Adding Classes to the Service Development Project

If a C/C++ service requires additional classes to compile, you must add them to the following locations:

- Service Development Project in the local workspace so that Designer can compile the service.
- Integration Server so that the server can compile the service. Designer does not automatically propagate classes that you add to the Service Development Project to Integration Server; you must add them to Integration Server manually. For more information about adding classes to Integration Server, see information about managing IS packages and how Integration Server stores IS package information in *webMethods Integration Server Administrator's Guide*.

Keep the following points in mind when adding classes to the Service Development Project:

- You add individual class files to the "classes" folder of the Service Development Project.
- If you have Java classes that are packaged together in jar files, you add the jar files to the "lib" folder of the Service Development Project.
- If you have set up IS package dependencies for a C/C++ service and if the service requires classes or jar files in these IS packages to compile, you must manually add the classes or jar files to Service Development Project.

To add classes and jar files to the Service Development Project

1. Open the Project Explorer view.
2. Expand the Service Development Project for the C/C++ service.
3. If you want to add class files to the Service Development Project, drag them from the file system into the "classes" folder of the Service Development Project in the Project Explorer view.

   When adding class files, ensure that you keep the structure of the Java package intact. For example, if you want to add com.accounting.orders.statusClass.class, you must first create the "com", "accounting", and "orders" folders within the "classes" folder as shown below:

```
- classes
  - com
    - accounting
      - orders
```

Then add the statusClass.class file to the “orders” folder.

Important:
Do not maintain the Java source files for these classes within the Service Development Project.

4. If you want to add jar files to the Service Development Project, drag them from the file system into the “lib” folder of the Service Development Project in the Project Explorer view.

   If you have the **Build automatically** Workspace preference selected, after adding new class and/or jar files to the Service Development Project, Designer automatically rebuilds the project. If you have the **Build automatically** preference turned off, you can force a rebuild by selecting **Project > Build Project**. You set the **Build automatically** preference using **Window > Preferences > General > Workspace**.

   After the project is rebuilt, Designer removes the compilation errors, if any, from the Problems view. However, the errors might still exist for the Folder class that resides in Integration Server.

   To correct the error, first, ensure that Integration Server has access to the required class and jar files. Then, open the C/C++ service in the Designer and save it again to force the compilation of the service on Integration Server.

Building the C/C++ Source Code

When you create a C/C++ service, Designer generates a source code file and a make file and places these files in the following directory:

Integration Server_directory \instances*instance_name*\packages\packageName\code\source

The names of the files will match the service name you specified in Designer. The source code file will be named <*servicename*>.c and the make file will be <*servicename*>.mak.

You create the C/C++ program in the *serviceName*Impl.c file, not the original file. The *serviceName*Impl.c file is the file in which the make file expects to find your source code. This step is taken to maintain a copy of the original source file to which you can refer, or revert to, during the development process.

To build the C/C++ source code

1. Locate the source code and make files. The source code file will be named <*servicename*>.c and the make file will be <*servicename*>.mak.

2. Copy the source code file to a new file (in the same directory) with the following file name:

   *serviceName*Impl.c

For example, if your service name is `PostPO`, you would create a copy of `PostPO.c` and name it `PostPOImpl.c`.

3. Edit the *`serviceName`*`Impl.c` file as necessary to build your service.

   This file contains instructive comments that will guide the development process. You can also refer to *webMethods Integration Server C/C++ API Reference* for information about how to use the webMethods C/C++ API to make the data in your service available to other services.

4. Edit the make file to customize it for your development environment. Set the following path settings:

Set...	To...
JDKDIR	The directory that contains the Java Development Kit.
SEVRDIR	The directory in which webMethods Integration Server is installed.

Important:
The source code file *`serviceName`*`.c` contains code based on the specification you used to define the inputs and outputs for the service. If you edit the specification, you need to regenerate the source code file. Designer does not update the *`serviceName`*`.c` file automatically. For more information about generating source code files for a C/C++ service, see "Creating a C/C++ Service" on page 351.

5. After you finish coding your service, run your make file to compile it. Following is a typical make command:

   ```
   make -f SalesTax.mak
   ```

   The make file compiles your program and puts the finished DLL in the code\libs directory in the package in which the service resides. If this directory does not exist when you run the make file, your program will not compile successfully.

6. Once your program compiles successfully, restart Integration Server to reload the code\libs directory. This makes the service available for execution and allows you to test it with Designer. For details on testing, see "Debugging C/C++ Services" on page 360.

Compiling the C/C++ Source Code

When you save a C/C++ service, Designer automatically compiles the C/C++ service in the Service Development Project and on Integration Server.

Keep the following points in mind when you are compiling a C/C++ service:

- You must add any additional Java classes that the C/C++ service requires to both the Service Development Project and to Integration Server. For more information, see "Adding Classes to the Service Development Project" on page 355. For more information about adding classes to

Integration Server, see information about managing IS packages and how Integration Server stores IS package information in *webMethods Integration Server Administrator's Guide*.

- Make sure you have a C/C++ compiler installed on the host where Integration Server is installed.
- When compiling the C/C++ service locally, Designer uses the default Java compiler settings. You can update these settings by updating the Service Development Project's Java Compiler properties.
- By default, the Service Development Project uses the default JRE that is configured for Designer. You can configure a different JRE using the Service Development Project's Java Build Path properties and setting a new JRE on the **Libraries** tab.

To compile a C/C++ service

1. If the service is not open in the C/C++ service editor, open it by double clicking the C/C++ service in the Package Navigator view.

2. Select **File > Save** to save and compile the C/C++ service.

Designer displays compilation errors from compiling the service in:

- **Problems view** for compilation errors from locally compiling the service

  If Problems view is not already open, you can open it by selecting **Window > Show View > Problems**.

  To view the line of code that caused the error, double click on the error in the Problems view. Designer shifts focus to the C/C++ service editor, with the cursor positioned at the line of code that caused the error.

- **Compiler Messages window** for compilation errors from Integration Server

  Designer writes the error messages from the server to the Designer log file, making them visible within the Error Log View.

  If you receive errors because the Java compiler cannot be found in Integration Server, ensure you have a Java compiler installed on the same machine as Integration Server and that you have added the location of the Java compiler to the system path.

Performance When Compiling a C/C++ Service

When Designer compiles the service locally, by default, it also rebuilds other classes in the Service Development Project. If you notice slower performance when you save the service, you can prevent Designer from rebuilding the other classes by updating the workspace preferences. Note that sometimes it is only the first attempt to save a C/C++ service that takes a long time and future compilations might go faster.

If you want to turn off the rebuild of other classes in the Service Development Project, select **Window > Preferences > General > Workspace** and clear the **Build Automatically** check box.

This preference affects all projects in the workspace. If you turn off automatic builds, you can manually force a build by selecting **Project > Build Project**.

Generating Code a C/C++ Service Can Use to Invoke a Specified Service

You can have Designer generate the code that invokes a selected service, which you can then add to a C/C++ service. Designer generates code that:

- Creates a Values object based on the service's declared input parameters
- Invokes the selected service passing it the Values object and catching exceptions
- Retrieves the returned Values object from the selected service
- Uses the key/value pairs in the returned Values object to assign variables based on the declared output parameters of the selected service

You do not have to generate code for all the input and output parameters. You can select to generate code for only the input parameters, only the output parameters, or you can select one or more input/output parameters for which to generate code.

When Designer generates code from the service input/output parameters, it puts the code on the clipboard. From there, you can paste it into a C/C++ service and modify it as necessary.

To generate code to invoke a service

1. In the Package Navigator view, open the service that you want to invoke.
2. If you want to generate code for a subset of the input/output parameters, on the **Input/Output** tab, select the parameters for which you want to generate code. To select more than one variable, press the CTRL key as you select parameters.
3. In the editor, right click the service to view the context menu, and select **Generate Code**.
4. In the Code Generation window, select **For calling this service from another service** and click **Next**.
5. For **Specification**, select the **Input** and/or **Output** check boxes to reflect the parameters for which you want to generate code.
6. For **Which Fields?** select one of the following:
   - **All Fields** if you want to generate code for all of the parameters identified by your **Specification** selection.
   - **Selected Fields** if you want to generate code for only the parameters you selected before starting the code generation.

7. Click **Finish**. Designer generates code and places it on the clipboard.

8. Paste the contents of the clipboard into a C/C++ service.

Debugging C/C++ Services

A C/C++ service is a Java service that calls the C program that you have created. Designer generates the Java code needed to successfully call your C program. In Designer, the primary way to debug a C/C++ service is to debug the Java class associated with the C/C++ service that Designer maintains in a Service Development Project.

When debugging a C/C++ service in this way, you can debug the primary method and shared code of the Java class that represents the C/C++ service. To debug the Java class, you launch it in debug mode and use the JDT debugger to suspend/resume the execution of the Java class, inspect variables, and evaluate expressions.

The actions you take when debugging a C/C++ service are:

- **Optionally set breakpoints** to identify locations where you want the debugger to suspend execution when running the Java class in debug mode. For more information, see "How to Suspend Execution of a Java Class while Debugging" on page 462.
- **Generate a test harness,** which is a Java class that you generate for the C/C++ service you want to debug. The logic that Designer generates for the test harness sets up the inputs, invokes the Java class, and displays the outputs.
- **Optionally create a Java Application launch configuration** to configure settings for debugging the Java class. For example, you might want to set JVM arguments to match the settings Integration Server uses so that your test more closely matches how the C/C++ service would execute in Integration Server. For more information, see "About Java Application Launch Configuration" on page 459. If you do not create a launch configuration, Designer creates one on the fly and saves it locally in an unexposed location of your workspace.
- **Launch the test harness in debug mode.** The test harness prompts for input values and then launches the Java class you want to debug in debug mode.

  By default, the debugger executes the Java class using the JRE in the Service Development Project where the C/C++ service resides. You can change the Service Development Project's JRE by updating the project's Java Build Path property. You can also specifically identify the JRE to use for debugging by identifying the JRE in the Java Application launch configuration.

  If the Java class being debugged invokes a service, the invoked service runs in Integration Server. The debugger treats the statement to invoke a service like any executable line of code in the Java class; that is, you can Step Over it and see results from it. You cannot use the debugger to Step Into the invoked service.

  If the debugger suspends execution of the service, Designer switches to the Debug perspective. The Debug view will show the test harness class and be positioned at the statement where the execution was suspended. You can use the other views in the Debug perspective to inspect the state of the C/C++ service to this point. You can use the actions in the Debug view toolbar

to resume the execution. For more information about suspending execution, see "How to Suspend Execution of a Java Class while Debugging" on page 462.

When the execution of the C/C++ service completes, the debugger displays a window that contains the service results.

For more information about debugging the C/C++ service by debugging its Java wrapper, see "Debugging Java Services" on page 455.

18 Building Services from .NET Methods

- Environment Setup for Creating .NET Services 364
- .NET Service Editor 364
- Creating a .NET Service 366
- Modifying the .NET Assembly Information 367
- Modifying the Class Lifetime for a .NET Service 368
- Running a .NET Service in Designer 369

A *.NET service* is a service that calls methods imported from .NET assemblies. Designer provides the .NET service editor for creating, viewing, and editing .NET services in your IS package.

Environment Setup for Creating .NET Services

The following system requirements must be met before using Designer to create .NET services:

- The webMethods Microsoft Package must be installed on the same Integration Server where the .NET services will reside. The webMethods Microsoft Package is composed of the following elements:
  - webMethods Microsoft Package, which is an Integration Server package
  - webMethods for Microsoft Code Generator Package, which is an Integration Server package
  - webMethods add-in for Microsoft Visual Studio .NET
- The Microsoft .NET Framework, which includes the Common Language Runtime (CLR), must be installed.

Make sure to install the webMethods Microsoft Package and the Microsoft .NET Framework on the same computer as the instance of Integration Server to which Designer is connected.

For more information about using the Microsoft .NET application platform with webMethods components and how to install the webMethods Microsoft Package, see the *webMethods Package for Microsoft .NET Installation and User's Guide* .

.NET Service Editor

Use the Designer .NET service wizard to create .NET services. After you have created a .NET service, use the .NET service editor to view and/or edit the service. If you attempt to edit a .NET service that you do not have locked for edit, you can still open it in the .NET service editor. However, the .NET properties, inputs, and outputs will be read only.

The .NET service has three tabs:

- **.NET Properties** tab contains information about the specific .NET method that a service calls. For more information, see ".NET Properties Tab" on page 365.
- **Input/Output** tab contains the input and output signature of the .NET service. For more information about declaring the input and output parameters for a service, see "About the Service Signature" on page 159.
- **Logged Fields** tab indicates the input and output parameters for which Integration Server logs data. For more information about logging the contents of input and output fields, see "Logging Input and Output Fields" on page 185.
- **Comments** tab contains the comments or notes, if any, for the .NET service.

Note:
You can use the Designer .NET editor to edit .NET services that you created in Developer.

.NET Properties Tab

The **.NET Properties** tab of the .NET service editor contains information about the specific .NET method that the service calls:

Property	Description
Domain Name	The name of the application domain in which the .NET service is to run.
Assembly Path	The location of the directory that holds the .NET assembly in which the method called by the .NET service resides. For information about changing this property, see "Modifying the .NET Assembly Information" on page 367. **Note:** Changes you make to this property apply to all .NET services that call methods in the same class definition.
Assembly Name	The name of the .NET assembly in which the method called by the .NET service resides. For information about changing this property, see "Modifying the .NET Assembly Information" on page 367. **Note:** Changes you make to this property apply to all .NET services that call methods in the same class definition.
Domain Configuration File	The configuration file associated with the domain. The file must be located in the assembly path. Enter only the file name. For more information about domain configuration file, see the *webMethods Package for Microsoft .NET Installation and User's Guide* .
Class Name	The name of the class that owns the method called by the .NET service. This field is read-only.
Class Lifetime	How Integration Server maintains the instance data for the class that owns the method called by the .NET service. A brief description of each setting follows. For a more detailed description of each and instructions for updating this property, see "Modifying the Class Lifetime for a .NET Service" on page 368.

Class Lifetime	Description
Global	Integration Server maintains a single version of the instance data that it is uses for all users that invoke a service associated with the class.

Property	Description	
	Session	Integration Server maintains separate versions of the instance data for individual users and uses the instance data when a user invokes a service associated with the class.
	Single-Use	Integration Server uses the instance data for only a single invocation of a service associated with the class.
	Static	Integration Server does not maintain instance data.
Class Timeout (Mins)	The number of minutes that Integration Server maintains instance data for the class that owns the method called by the .NET service. This property is only available when you select **Session** for the **Class Lifetime** field. For more information about setting the **Class Timeout** property, see "Modifying the Class Lifetime for a .NET Service" on page 368.	
Method Name	The name of the method called by the .NET service. This field is read-only.	

Creating a .NET Service

You create a .NET service using the .NET service wizard. Using the wizard, you scan existing .NET assemblies to determine the methods they contain. You can then import the methods into Designer. Using the Designer .NET service editor, you can create .NET services that call those methods. For more information about the editor, see ".NET Service Editor" on page 364.

Before you can create a .NET service, make sure the .NET Common Language Runtime (CLR) is loaded (i.e., started). If it is not, use Integration Server Administrator to load it. For instructions, see *webMethods Package for Microsoft .NET Installation and User's Guide* .

You can invoke .NET services from flow services. You can also execute them from Designer. For more information, see "Running a .NET Service in Designer " on page 369.

To create .NET services from a method

1. In the Service Development perspective, select **File > New > .NET Service**.
2. In the Create New .NET Service wizard, expand the IS package in which you want the new service(s) to reside and select the folder in which you want to create the service(s).
3. Click **Next**.
4. In the Assemblies for Auto Conversion panel, locate a .NET assembly on a drive where the CLR can access it.

This panel depicts the white-listed directories. The only files displayed in this dialog box are assembly DLLs or EXEs. You can select multiple .NET assemblies for importation.

5. After you have selected assemblies, click **Next**.

6. In the Select Specific .NET Services panel, select the methods you want to import into Designer. By default, all methods in all selected assemblies are selected. You can select or clear whole assemblies, whole classes, or individual methods as needed.

7. Click **Finish**. Designer creates a .NET service for each method and places it in the specified folder.

For assemblies that are not located in the same windows domain or on the same machine as the CLR, the CLR might fail to load an assembly and issue a security error if the assembly was compiled with the unsafe option or if user permissions for the remote directory do not permit access. To resolve either condition, copy the assembly to the machine where the CLR resides, set directory permissions appropriately, or configure a trust relationship between the domains.

You can view the variables for the resulting .NET service in the **Input/Output** tab. In addition to the variables supported by a specific method, there are standard variables that are part of each .NET service.

Modifying the .NET Assembly Information

When you create a .NET service to call a Microsoft .NET method, Integration Server stores information that the .NET CLR needs to load the method into its processing space. If you change the location or name of the .NET assembly in which the method resides, use the **.NET Properties** tab to modify the information so that Integration Server can continue to call the method. If you do not update the information, attempts to call that method from Integration Server will fail.

Note:
When you create multiple .NET services from an assembly, as described in "Creating a .NET Service" on page 366, all the services share information about the assembly. When you change shared information for one .NET service, Integration Server changes the information for all .NET services associated with the assembly.

To change information about a .NET method

1. In the Service Development perspective, open the .NET service by double clicking it in the Package Navigator view.

2. Click the **.NET Properties** tab.

3. Perform one or more of the following actions:

To...	Do this...
Change the assembly path name	In the **Assembly Path** field, type the new location of the directory that holds the .NET assembly in which the method resides.
Change the assembly name	In the **Assembly Name** field, type the new name of the .NET assembly in which the method resides.
Change the domain name	In the **Domain Name** field, type the new domain name.
Change the domain configuration file	In the **Domain Configuration File** field, type the name of the new domain configuration file.

4. Select **File > Save**.

5. Stop and restart the CLR to clear the cache and make sure the correct assembly is loaded.

6. Reload the webMethods Microsoft Package (WmDotNet package) in Integration Server Administrator.

   For instructions for how to stop and restart the CLR, see *webMethods Package for Microsoft .NET Installation and User's Guide* .

Modifying the Class Lifetime for a .NET Service

The **Class Lifetime** property indicates how you want Integration Server to maintain instance data for a .NET class running on the CLR. The instance data is a set of variables associated with the .NET class. When a .NET service executes, the CLR uses the instance data for the class that owns the method called by the .NET service.

Note:
When you set the **Class Lifetime** property for a service, Designer automatically sets the **Class Lifetime** property of all .NET services associated with the same class to the same setting.

To modify the Class Lifetime for a .NET Service

1. In the Service Development perspective, open the .NET service by double clicking it in the Package Navigator view.

2. Click the **.NET Properties** tab.

3. Set the **Class Lifetime** property to indicate how you want Integration Server to maintain instance data for a class.

Class Lifetime	Description
Global	Integration Server creates a single instance of the class, or object, which has an unlimited lifetime. The class shares instance data among all sessions. You can create only one instance of a global object of a given type. Use this setting when you want Integration Server use the same instance data for all users in multiple sessions.
Session	Integration Server creates a separate object for each user. The object exists until the user session is closed or until the object times out. The default timeout value for the object is three minutes. Use the **Class Timeout** property to specify a different timeout value for an object.
Single-Use	Integration Server creates and destroys an object each time a .NET service calls a method in the class. Instance data might be kept during the lifetime of the object.
Static	Integration Server does not create an object to save instance data. All methods of the class are static. Use this setting when the .NET service calls a method that does not require any session data to be kept.

4. If you set the lifetime to **Session**, specify a value for the **Class Timeout (Mins)** property to define the timeout value for objects.

   Set a high enough value so that Integration Server does not prematurely destroy objects under normal usage. The default is `3` (i.e., three minutes).

   Note: Integration Server starts counting the minutes for the timeout when an instance of the class is created. Whenever a .NET service accesses the class, Integration Server resets the count.

5. Select **File > Save**.

Important:
If you set the **Class Lifetime** property to **Global** or **Session**, the instance data can be used across multiple invocations of methods in a class. If multiple services are using a given global object or a session object at the same time, those objects need to be thread safe.

Running a .NET Service in Designer

1. In Package Navigator view, select the .NET service you want to run.

2. In Designer: **Run > Run As > Run Service**

3. If multiple launch configurations exist for the service, use the Select Launch Configuration dialog box to select the launch configuration that you want Designer to use to run the service.

4. If the launch configuration is set up to prompt the user for input values or there is no launch configuration, in the **Enter Input for *serviceName*** dialog box, specify input values for the service.

   a. In the *domainName* field, type the domain name of a new or existing application domain name on Integration Server.

   Note:
   If you do not specify an application domain, the service runs in the default webmDomain application domain.

   b. In the *marshallingType* list, select one of the following:

Value	Marshalling Type
refid	Reference ID
xml	XML marshalling

   c. If there are any other input fields, specify values for them.

5. Click **OK** to run the service.

19 Building XSLT Services

- What Is XSLT? 372
- What Is an XSLT Service? 372
- How Does an XSLT Service Work? 373
- What Is a Translet? 373
- About the XSLT Service Editor 374
- Overview of Building XSLT Services 374
- Creating an XSLT Service 374
- XSLT Service Signature 375
- Running an XSLT Service 378
- Debugging an XSLT Service 378
- Guidelines for the XSLT Style Sheet 379
- Using Name/Value Pairs with an XSLT Service 380
- Configuring XSLT Transformer Factory Settings 386

You can use Designer to create XSLT services that transform XML source data according to instructions in an associated style sheet.

What Is XSLT?

XSLT (EXtensible Stylesheet Language Transformations) is a language used to transform XML documents into other XML documents or formats. It is an industry standard for XML data mapping, based on its flexibility and reusability. Integration Server supports that flexibility by providing a straightforward mechanism for converting XML data within Designer.

What Is an XSLT Service?

An *XSLT service* transforms XML source data by applying the instructions in an associated document or style sheet. The XML data can be transformed into HTML for display as a web page, into plain text, or into different XML formats, depending on the XSLT code in the style sheet. You can create multiple XSLT services, each with its own style sheet, that define different types of transformations.

The XSLT rules for a specific XML transformation are stored in the service's style sheet. At run time, the service uses the XSLT style sheet to transform XML data passed to the service.

XSLT services support the following standards:

- Java API for XML Processing (JAXP)
- XSL Transformations (XSLT) Version 1.0
- XML Path Language (XPath) Version 1.0

An XSLT service is identified in the Package Navigator view of Designer with this icon: . If you rename, move, or delete an XSLT service, Designer automatically updates any references to it and renames the style sheet file to match the service name.

You can drag and drop an XSLT service into a flow service and call it as a step in the service, invoke it from a Java service, or invoke it directly using an HTTP request. The XSLT service must reside on the same server as the flow service.

Note:
XSLT services cannot be invoked by a trigger. Triggers pass documents to their associated services. Any input supplied by a trigger would be invalid because XSLT services use fixed input and output variables.

By default, Integration Server uses an interpretive processor to perform the transformation. But, to save processing time, you can instruct Integration Server to use a compiling processor to create a compiled style sheet or translet. The compilation processor runs just one time and produces a translet that can be reused during subsequent transformations.

To instruct Integration Server to use the compiling processor, you use the *useCompilingProcessor* input parameter of an XSLT service. For more information about *useCompilingProcessor*, see "XSLT Service Signature" on page 375.

Note:
You can use a compiling processor to create a translet only if you are using style sheets that conform to XSLT Version 1.0.

How Does an XSLT Service Work?

An XSLT service applies the rules in its XSLT style sheet to transform XML data.

When executed, the XSLT service calls an external XSLT engine to convert the XML data. The external XSLT engine must be Java API for XML Processing (JAXP)-compatible. By default, Integration Server includes the Xerces parser and the Xalan XSLT style sheet processor from the Apache Software Foundation. The Java archives for Xerces and Xalan are located in the *Software AG_directory* \common\lib\ext directory.

To use a more current version of the Xalan or other 3rd-party XSLT processor, you must place all the related jar files in the *Integration Server_directory* \instances*instance_name*\packages\WmXSLT\code\jars directory and configure the transformer factory settings to specify the appropriate XSLT transformer class. You must also identify a package dependency on the WmXSLT package for XSLT services in custom packages. For more information about identifying package dependencies, see "About Package Dependencies" on page 149.

Note:
Transforming a very large XML file can exceed the memory parameters set in Integration Server, resulting in the following error message: "Could not run filename. java.lang.reflect.InvocationTargetException: OutOfMemoryError". If this occurs, edit the wrapper.java.maxmemory property in the custom_wrapper.conf file. For information about changing the JVM heap size by editing the Java properties in the custom_wrapper.conf file, see the *webMethods Integration Server Administrator's Guide*.

What Is a Translet?

A *translet* is a compiled java class that you can use to perform XSL transformations. By default, Integration Server uses an interpretive processor to process a style sheet. But for greater efficiency, you can instruct Integration Server to use a compiling processor provided by Xalan. The compiling processor compiles the style sheet into a translet. The style sheet compilation is performed only once per style sheet (unless the style sheet is modified) and the resultant translet is reused during subsequent transformations. As a result, transformations are performed more quickly.

Integration Server writes the translet to the same folder that contains the associated style sheet. The translet will be available even after Integration Server restarts and can be used in subsequent transformations.

To instruct Integration Server to use the compiling processor, you use the *useCompilingProcessor* input parameter. For more information about *useCompilingProcessor*, see "XSLT Service Signature" on page 375.

About the XSLT Service Editor

When you create an XSLT service, Designer creates its associated style sheet. The style sheet is empty by default. You can type in XSLT code and build the style sheet from scratch, or you can import an existing style sheet as a basis for the new document. You use the **Source** tab in the XSLT style sheet editor in Designer to add or edit the contents of a style sheet. The style sheet editor includes standard editing functions, as well as features for supporting the XSLT programming language, such as highlighting certain elements in color.

The XSLT style sheet contains XSLT template rules and instructions for transforming XML data provided as input to the XSLT service. You can enter and modify the code in the service's style sheet using the XSLT service editor in Designer.

Designer provides the full capabilities of the Eclipse XSLT editor, for example, source formatting, code completion, undo and redo, find and replace etc.

You can also use the **Comments** tab to specify comments or notes, if any, for the XSLT service

You can use the **Window > Preferences > XML > XSL** page to customize the XSLT service editor. You can also access the preferences page by right-clicking inside the XSLT editor and selecting **Preferences**.

Overview of Building XSLT Services

The following are the basic tasks you perform to create an XSLT service:

Task 1	Create an XSLT service and the associated XSLT style sheet. For more information, see "Creating an XSLT Service" on page 374.
Task 2	Edit the XSLT style sheet and write the XSLT transformation code. Information about writing XSLT code is outside the scope of this help. However, for some suggestions for creating a well-formed style sheet, see "Guidelines for the XSLT Style Sheet" on page 379.

Creating an XSLT Service

When you create an XSLT service, you are creating a style sheet containing the XSLT transformation rules.

To create an XSLT service

1. In the Package Navigator view of Designer, select **File > New > XSLT Service**.

2. In the New XSLT Service dialog box, select the folder in which you want to save the service.

3. In the **Element name** field, type the name for the XSLT service. It would be helpful if you give the service a name that describes the type of transformation that the style sheet defines. Click **Next**.

4. Do one of the following to create the style sheet:

   - To create an empty XSLT style sheet, select **None**. This is the default.
   - To import the text of another XSLT file to use as a basis for this service's style sheet, select **XSLT file**. Click **Browse** to locate and select the file whose contents you want to import.
   - To use a template you want to use to initialize a default set of properties for the service, select **Use template** and select the appropriate template.

     Note: Designer lists the templates that are defined on the **Window > Preferences > XML > XSL > Templates** page.

5. Click **Finish**.

   Designer refreshes the Package Navigator view and displays the new service in the XSLT service editor.

   Designer saves the style sheet as a text file using the naming convention *serviceName*.xsl. It is stored in the same directory as the service's node.ndf file, that is, within the \ns directory of the package containing the service. For example, when you save the XSLT service com.example.inventory:convert, Designer names the style sheet file convert.xsl and stores it in the following directory: *Integration Server_directory* \instances*instance_name*\packages*packageName*\ ns*folderName*\com\example\inventory\convert.

   Important:
   Do not rename the style sheet file. When an XSLT service is executed, it looks in the service directory for a style sheet called *serviceName*.xsl that contains instructions for transforming the XML data. If the appropriately named file is not in that location, the service creates an empty style sheet file, and ignores the renamed one. However, you can rename an XSLT service; Designer automatically renames the style sheet file to match the new service name.

   You can specify service properties such as the run time settings, service retry, service auditing, and permissions using the Properties view. For more information, see "Building Services" on page 157.

XSLT Service Signature

All XSLT services use the same set of input and output parameters.

The standard input variables for the XSLT service are described below. These specify the type of XML input the service expects, as well as any name/value pairs.

Input Parameters

xmldata	**byte[]** Optional. XML to transform.
xmlUrl	**String** Optional. URL of the XML to transform.
filename	**String** Optional. Fully qualified name of the file that contains the XML to transform. The file must be located on the local machine.
xmlStream	**Input stream** Optional. XML to transform.
node	**com.wm.lang.xml.Node** Optional. Node that contains the XML to transform.
xslParamInput	**Document** Optional. Name/value pairs to pass to the style sheet. For instructions on setting up your style sheet to work with this input variable, see "Passing Name/Value Pairs from the Pipeline to the Style Sheet" on page 380.
encoding	**String** Optional. Default character set to use for encoding the data transferred during this session. Specify an IANA-registered character set (for example, `ISO-8859-1`). If you do not set *encoding*, the default JVM encoding is used.
$useCache	**String** Optional. Specifies whether to use the cached version of the style sheet or the most recent version. To use this parameter, add *$useCache* to the input pipeline of the XSLT service and set the value to `false` to always call the most recent version of the style sheet. **Important:** By default, the XSLT transformation engine caches style sheets, which improves performance significantly. However, if you are working on the XSLT service in a development environment and making frequent changes to the style sheet, it is convenient to always specify the most recent version when you call the XSLT service from a flow service.
useCompilingProcessor	**Boolean** Optional. Specifies whether or not to use the Xalan compiling processor (XSLTC), which creates and uses compiled style sheets or translets. Set to: ■ `true` to use the org.apache.xalan.xsltc.trax.TransformerFactoryImpl class as a transformer factory. If no translet currently exists for the style sheet, the processor creates one. If a translet exists and the style sheet has changed since the translet was created, the processor replaces the existing translet with a new one. If the style sheet has not changed since the translet was created, the processor reuses

the existing translet. This setting overrides the setting specified on the home page of the WmXSLT package.

- `false` to use the transformer factory that is specified on the home page of the WmXSLT package. The default is `false`.

Note:
The compiling processor is only supported by the Xalan processor that supports XSLT 1.0. Consequently, to create translets you must only use style sheets that are XSLT 1.0 compatible.

loadExternalEntities **String** Optional. Specifies whether or not to load external entities (file URIs, HTTP URLs, and so on) referenced in the XML that the service receives or in the XSLT style sheet the service uses to transform the XML. Set to:

- `true` to load content from all external entities that are referenced in the XSLT style sheet or in the XML. This is the default.
- `false` to not load content from external entities that are referenced in the XSLT style sheet or in the XML. Use this setting to prevent attacks from external entities by blocking those entities.

Important:
To help prevent an external entity attack in a production environment, set *loadExternalEntities* to `false` in each instance of the transformSerialXML service.

Output Parameters

results **String** String that contains the transformed XML.

xslParamOutput **Document** Conditional. Document that contains name/value pairs that were returned by the style sheet. This value is present only if you add name/value pairs to it from your style sheet. For more information about passing name/value pairs from the style sheet to the pipeline and for instructions on setting up your style sheet to work with this parameter, see "Using Name/Value Pairs with an XSLT Service" on page 380.

Usage Notes

The *xmldata*, *xmlUrl*, *filename*, *xmlStream*, and *node* input parameters are mutually exclusive. Use any one of these parameters to specify the type of XML input.

If the *loadExternalEntities* input parameter is set to false, you can have the service load, read, and transform content from a trusted external entity by doing one of the following:

- Place the trusted external entity file in the Integration Server installation directory or subdirectories.
- Include the trusted external entity in the list of trusted entities identified in the server parameter watt.core.xml.allowedExternalEntities. For more information about this parameter, see *webMethods Integration Server Administrator's Guide*.

If the *loadExternalEntities* input parameter is not specified in the service signature, Integration Server checks the value of the server parameter watt.core.xml.expandGeneralEntities. If this parameter is set to `false`, the service blocks all external entities that are not included in the list of trusted entities specified in watt.core.xml.allowedExternalEntities. For more information about watt.core.xml.expandGeneralEntities, see *webMethods Integration Server Administrator's Guide*.

Running an XSLT Service

When you run an XSLT service using **Run > Run As > Run Service**, Designer calls the service (just as an ordinary Integration Server client would) and receives its results. To run an XSLT service, you need to specify an XML file to use as input. The service executes once, from beginning to end (or until an error condition forces it to stop) on the Integration Server on which you have an open session. Integration Server validates the XSLT code in the style sheet at run time and displays an error message when it encounters invalid code. Results from the service are returned to Designer and displayed in the Results view.

For more information about debugging services, see "Running Services" on page 403.

Debugging an XSLT Service

You can debug an XSLT service using the Eclipse debugging framework using **Run > Debug Configurations** in Designer. In order to debug an XSLT service, you must first create a launch configuration. You can then debug the XSLT service using the re-usable launch configuration.

Note:
You cannot use the Eclipse debugging framework for XSLT services in which you have performed pipeline customizations. For more information about customizing an XSLT service in the pipeline, see "Passing Name/Value Pairs from the Style Sheet to the Pipeline" on page 382.

Creating a Launch Configuration for an XSLT Service

To debug an XSLT service in Designer, you create and then run a launch configuration. You can specify input values for debugging the XSLT service in the launch configuration. Use the following procedure to create a re-usable launch configuration that you can run to debug an XSLT service.

To create a launch configuration for debugging an XSLT service

1. In the Service Development perspective of Designer, select **Run > Debug Configurations**.

2. On the Configurations tree, select **XSL** and click .

3. In the **Name** field, specify a name for the new launch configuration.

4. In the **XML Input File** field, select a source XML file.

5. Under **Transformation Pipeline**, click **Add Files**. Browse to your local workspace to select the .xsltservice source file corresponding to the XSLT service that you want to debug.

6. Specify transformation parameters if required.

7. Click **Apply**.

8. Click **Debug** to debug the XSLT service now. Otherwise, click **Close**.

Debugging an XSLT Service

To debug an XSLT service using a launch configuration

1. In the Service Development perspective of Designer, select **Run > Debug Configurations**.

2. On the Configurations tree, select **XSL** and open the launch configuration that you want to run to debug the XSLT service.

3. Click **Debug** to debug the XSLT service using this launch configuration.

Guidelines for the XSLT Style Sheet

The XSLT style sheet contains the XSLT rules and instructions that the XSLT service applies to the provided XML. Some suggestions for creating a well-formed style sheet are:

- Use the XSL style sheet element as the topmost element. For example:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <! -- XSLT code goes here -->
</xsl:stylesheet>
```

  This explicitly defines the XSLT namespace and version.

- Use the `xsl:` prefix for all standard XSLT elements as defined in the http://www.w3.org/1999/XSL/Transform conventions. The prefix prevents conflicts between the standard XSLT elements and any locally defined elements that have the same name. For example, without the prefix to distinguish them, the `<xsl:key attribute="value">` element, used for cross-referencing data, could be confused with a `<key>` element defined for a hardware business.

- Use valid XSLT code. Integration Server validates the code at run time.

Using Name/Value Pairs with an XSLT Service

The XML data passed to an XSLT service for transformation might contain elements that include a name/value pair, which consists of an XML attribute and its corresponding value. For example, in the XML element `<customer custid=”wm339”>, custid=”wm339”` is a name/value pair.

Using the appropriate instructions in the XSLT style sheet in conjunction with the XSLT service *xslParamInput* parameter, you can:

- Override the value of a name/value pair defined in the style sheet. By passing a new value from the pipeline to the style sheet you can specify values during the transformation that were not available when you wrote the style sheet, and run different transformations without changing the underlying XSLT style sheet.
- Define a new name/value pair in the style sheet, and pass it to the pipeline when you run the service.

Passing Name/Value Pairs from the Pipeline to the Style Sheet

An XSLT service can pass name/value pairs from the pipeline to the style sheet to use while transforming the XML data.

First, using the *xslParamInput* input variable, specify a new value for the name/value pair you want to override. Then, define the name/value pair as an XSLT parameter in the XSLT style sheet. An XSLT parameter is an XSLT element similar to a variable, except that its value can be changed at the time the XSLT style sheet is processed.

At run time, the value of the XSLT parameter defined in the style sheet is replaced with the new value defined in the *xslParamInput* variable. The tasks that you need to perform to pass name/value pairs from the pipeline to the style sheet are:

Task 1	In the Pipeline, specify new values for each name/value pair you want to override in the style sheet. For more information about specifying new values for each name/value pair you want to override in the style sheet, see “Specifying New Values for Name/Value Pair” on page 380.
Task 2	Define each name/value pair as an XSLT parameter in the style sheet. For more information about defining each name/value pair as an XSLT parameter in the style sheet, see “Defining Name/Value Pair as an XSLT Parameter” on page 381.

Specifying New Values for Name/Value Pair

To specify new values for each name/value pair you want to override in the style sheet

1. Open the flow service that contains the XSLT service as a flow step.

2. In the flow service editor, select the INVOKE step that contains the XSLT service you want to alter.

3. Select the *xslParamInput* parameter of the XSLT service in the pipeline.

4. Using the **Enter Input for *variableName*** dialog box, enter the name of the XML attribute in the **Name** field.

5. In the **Value** field do one of the following to supply the new value.

   - If you want to assign a literal value to the variable, type that value. The value must be of the same data type as the variable.
   - If you want to derive the value from a String variable in the pipeline, type the name of that variable enclosed in % symbols (for example, %Phone%). Then, select the **Perform pipeline variable substitution** check box.
   - If you want to derive the value from a global variable, type the name of that global variable enclosed in % symbols (for example, %myFTPServer%. Then, select the **Perform global variable substitution** check box.

   Note:
   If a pipeline variable and a global variable have the same name and you select both the **Perform global variable substitution** and **Perform pipeline variable substitution** check boxes, Integration Server uses the value of the pipeline variable.

6. If you want Integration Server to use the specified value only if the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server will always apply the specified value.)

7. Click **OK**.

Defining Name/Value Pair as an XSLT Parameter

To define each name/value pair as an XSLT parameter in the style sheet

1. In the service's XSLT style sheet, use the `xsl:param` element to define a corresponding XSLT parameter for each name/value pair you defined using *xslParamInput*. For example:

```
<xsl:param name="name1"/>
<xsl:param name="name2"/>
<xsl:param name="name3"/>
.
.
.
```

```
<xsl:param name="nameN"/>
```

2. At run time, the XSLT service will pass the new value from the pipeline to the style sheet. The style sheet will use the new value during the transformation of the XML data.

Passing Name/Value Pairs from the Style Sheet to the Pipeline

You can write an XSLT style sheet that will create new name/value pairs and pass them to the service to put in the pipeline. You do this using an *XSLT extension*, which is an element that is written to support a particular function and that is not part of the standard XSLT specification.

Use the XSLT extension mechanism method named IOutputMap.put(*Object name*, *Object value*) to add name/value pairs to an output object of type IOutputMap. The IOutputMap interface (com.wm.pkg.xslt.extension.IOutputMap) defines method put().

For more information about the extension, see the webMethods WmXSLT Package Java API Reference, located in the *Integration Server_directory* \instances*instance_name*\packages\WmXSLT\pub\doc\api directory.

To pass name/value pairs from the XSLT style sheet to the pipeline

1. Open the XSLT style sheet.

2. Identify the IOutputMap put() method as an extension function by declaring a namespace that corresponds to the IOutputMap interface. Do this by adding the following namespace attribute within the `xsl:stylesheet` element:

```
xmlns:IOutputMap="com.wm.pkg.xslt.extension.IOutputMap"
```

3. Define an XSLT parameter named *`$output`* in the style sheet, as follows:

```
<xsl:param name="output"/>
```

Note:
After you define an XSLT parameter, you identify it to the XSLT processor as a variable, rather than text, by prefixing the name with a dollar sign.

Internally, the *$output* parameter is of type com.wm.pkg.xslt.extension.IOutputMap.

If you are using the XALAN compiling processor, you must use the xsltc:cast function to explicitly cast the $output object into com.wm.pkg.xslt.extension.IOutputMap. For example:

```
 <xsl:variable name="outputVariable"
select="xsltc:cast('com.wm.pkg.xslt.extension.IOutputMap', $output)"/>
```

4. For each new name/value pair you want to add to the *`$output`* parameter, insert the following `xsl:value-of` element, where *`key`* identifies a name/value pair and xpath is any valid XPATH expression:

```
<xsl:value-of select="IOutputMap:put($output,'key',string(xpath))"/>
```

The style sheet passes the contents of the *$output* parameter to the *xslParamOutput* variable of the service, and the service puts the resulting document in the pipeline.

If you are using the XALAN compiling processor, you must use the *outputVariable* variable (described in the previous step) when adding name/value pairs to the output. For example:

```
<xsl:value-of select="IOutputMap:put($outputVariable,'key',
string(xpath))"/>
```

Sample Style Sheet: Adding Name/Value Pairs to the Pipeline

The XSLT style sheet below uses the IOutputMap interface and extension functions from the Java classes Date and IntDate to add name/value pairs to the pipeline. IntDate is a simple class that converts a set of integers into a Date object.

```
<?xml version="1.0" ?>
<!--Declares namespaces for the XSL elements and Java functions-->
<xsl:stylesheet version="1.1"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:IOutput="com.wm.pkg.xslt.extension.IOutputMap"
    xmlns:Date="java.util.Date"
    xmlns:IntDate="com.wm.pkg.xslt.samples.date.IntDate"
    exclude-result-prefixes="IOutput Date IntDate">
<!--Defines the XSLT parameters-->
    <xsl:param name="output"/>
    <xsl:param name="year"/>
    <xsl:param name="month"/>
    <xsl:param name="day"/>
    <xsl:param name="hour"/>
    <xsl:param name="minute"/>
    <xsl:param name="second"/>
    <xsl:param name="date" select=
"IntDate:getDate($year,$month,$day,$hour,$minute,$second)"/>
    <xsl:output method="xml" indent="yes" />

    <xsl:template match="/" >
    <!--Converts the results of each parameter to a text string and adds
        it to the $output variable-->
        <xsl:value-of select="IOutput:put($output, 'year', $year)" />
        <xsl:value-of select="IOutput:put($output, 'month', $month)" />
        <xsl:value-of select="IOutput:put($output, 'day', $day)" />
        <xsl:value-of select="IOutput:put($output, 'hour', $hour)" />
        <xsl:value-of select="IOutput:put($output, 'minute', $minute)" />
        <xsl:value-of select="IOutput:put($output, 'second', $second)" />
        <xsl:value-of select="IOutput:put($output, 'date', $date)"/>
        <xsl:apply-templates />
    </xsl:template>


    <!--  Adds a new element with a matching name for each text string
          in the result tree.-->
    <xsl:template match="*">
        <xsl:element name="{name()}">
            <xsl:apply-templates />
        </xsl:element>
```

```
    </xsl:template>

    <xsl:template match="date" xml:space="preserve">
    <!--For each variable, produces a text string consisting of a label
        for the variable followed by the value of the variable-->
      <xsl:text>year   : </xsl:text><xsl:value-of select="$year"/>
      <xsl:text>month  : </xsl:text><xsl:value-of select="$month"/>
      <xsl:text>day    : </xsl:text><xsl:value-of select="$day"/>
      <xsl:text>hour   : </xsl:text><xsl:value-of select="$hour"/>
      <xsl:text>minute : </xsl:text><xsl:value-of select="$minute"/>
      <xsl:text>second : </xsl:text><xsl:value-of select="$second"/>

      <!--Invokes a Date function and prints the resulting values-->
      <xsl:text>converts to</xsl:text>

      <xsl:value-of select="Date:toString($date)"/>
  </xsl:template>
</xsl:stylesheet>
```

The following sample shows the IntDate class updated to use the compiling processor.

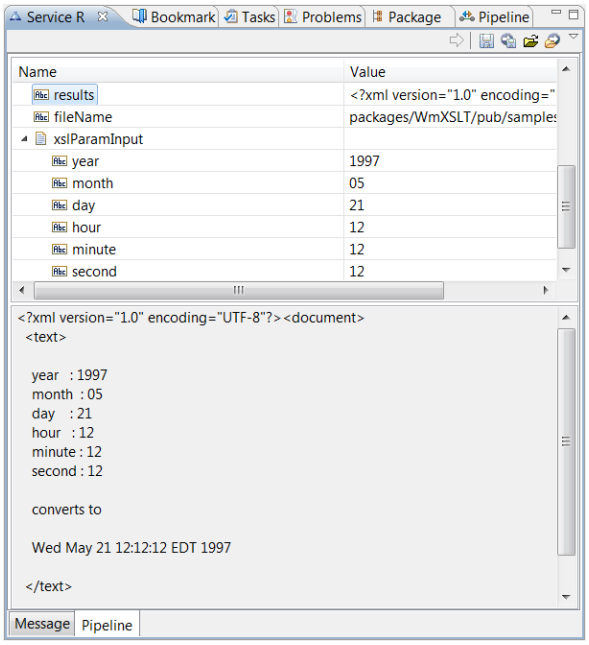
```
<?xml version="1.0" ?>
<!--Declares namespace for the XSL elements and Java functions-->
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsltc="http://xml.apache.org/xalan/xsltc"
  xmlns:IOutput="com.wm.pkg.xslt.extension.IOutputMap"
  xmlns:Date="java.util.Date"
  xmlns:IntDate="com.wm.pkg.xslt.samples.date.IntDate"
  exclude-result-prefixes="IOutput Date IntDate">

  <!--Defines the XSLT parameters -->
  <!--Explicitly casts $output-->
  <xsl:param name="output" select=
"xsltc:cast('com.wm.pkg.xslt.extension.IOutputMap', $output)" />
  <xsl:param name="year"/>
  <xsl:param name="month"/>
  <xsl:param name="day"/>
  <xsl:param name="hour"/>
  <xsl:param name="minute"/>
  <xsl:param name="second"/>
  <xsl:param name="date" select=
 "IntDate:getDate($year,$month,$day,$hour,$minute,$second)"/>
  <xsl:output method="xml" indent="yes" />

  <xsl:template match="/" >
    <!--Converts the results of each parameter to a text string and adds
        it to the $output variable-->
    <xsl:value-of select="IOutput:put($output, 'year', $year)" />
    <xsl:value-of select="IOutput:put($output, 'month', $month)" />
    <xsl:value-of select="IOutput:put($output, 'day', $day)" />
    <xsl:value-of select="IOutput:put($output, 'hour', $hour)" />
    <xsl:value-of select="IOutput:put($output, 'minute', $minute)" />
    <xsl:value-of select="IOutput:put($output, 'second', $second)" />
    <xsl:value-of select="IOutput:put($output, 'date', $date)" />
     <xsl:apply-templates />
  </xsl:template>

   <!--Adds a new element with a matching name for each text string in
       the result tree-->
```

```
    <xsl:template match="*">
      <xsl:element name="{name()}">
        <xsl:apply-templates />
      </xsl:element>
    </xsl:template>


 <xsl:template match="date" xml:space="preserve">
   <!--Produces text string consisting of label for variable followed
        by value of variable-->
  <xsl:text>year   : </xsl:text><xsl:value-of select="$year"/>
        <xsl:text>month  : </xsl:text><xsl:value-of select="$month"/>
        <xsl:text>day    : </xsl:text><xsl:value-of select="$day"/>
        <xsl:text>hour   : </xsl:text><xsl:value-of select="$hour"/>
        <xsl:text>minute : </xsl:text><xsl:value-of select="$minute"/>
        <xsl:text>second : </xsl:text><xsl:value-of select="$second"/>

  <!--Invokes Date function and prints resulting values-->
    <xsl:text>converts to</xsl:text>
    <xsl:value-of select="Date:toString($date)"/>
    </xsl:template>
</xsl:stylesheet>
```

When you run the service, provide the following input:

filename	packages/WmXSLT/pub/samples/xdocs/date.xml	
xslParamInput	year	1997
	month	05
	day	21
	hour	12
	minute	12
	second	12

The service uses the style sheet to transform the XML data from the source file specified in *filename*, converts the data into an XML string, and puts the string in the pipeline as the *results* parameter. The service also generates the name/value pairs and puts them in the xslParamOutput document. The Results view will display results similar to the following:



Configuring XSLT Transformer Factory Settings

You can use Integration Server Administrator to specify which transformer factory you want Integration Server to use.

To configure transformer factory settings

1. Open Integration Server Administrator if it is not already open.
2. In the **Packages** menu of the Navigation panel, click **Management**.
3. In the **Package List**, find the row for the WmXSLT package and click the home icon for the package.
4. Click **Edit Transformer Factory Settings**.
5. From the **Transformer Factory** list, select a factory or select **Other** and type in the name of the factory you want to use.
6. In the **Transformer Factory Attributes** field, optionally enter any factory attributes you want to specify. The attributes must take the form *`attribute=value`*. You can only specify attributes that accept strings or boolean values. For more information about possible attributes, refer to the documentation from your XSLT processor vendor.
7. Click **Save Changes**.

20 Managing Cloud Connector Services

- Creating a Cloud Connector Service 388
- Editing a Cloud Connector Service for a SOAP-Based Provider 389
- Editing a Cloud Connector Service for a REST-Based Provider 394

You create and manage cloud connector services using Designer.

Keep the following points in mind when creating a cloud connector service:

- Before you create a cloud connector service, ensure that the CloudStreams connector associated with your desired cloud application provider is installed. Also ensure that a cloud connection pool is created for that connector.
- If you are working with a SOAP-based provider, you should create at least one cloud connector service for each operation defined in the cloud connector descriptor. The operations contain a reference to a SOAP operation, defined in the connector's WSDL. For more information about SOAP operations, see "Editing a Cloud Connector Service for a SOAP-Based Provider" on page 389.
- If you are working with a REST-based provider, you should create at least one cloud connector service for each REST resource. For more information about REST resources, see "Editing a Cloud Connector Service for a REST-Based Provider" on page 394.

Note:
Information about cloud connector services is located in *webMethods Service Development Help* and also in the documentation specific to your CloudStreams provider (for example, *webMethods CloudStreams Provider for Salesforce.com Installation and User's Guide*). Both documents include the "Managing Cloud Connector Services" on page 387 topic, which provides information about how to create a cloud connector service for the operations defined in the WSDL of the CloudStreams provider or for the REST resources, using Designer. The documentation specific to your CloudStreams provider also contains information about how to create a cloud connection and configure the session management parameters, using the Integration Server Administrator.

Creating a Cloud Connector Service

You create a cloud connector service using Designer.

To create a cloud connector service

1. Open the **Service Development** perspective in Designer if it is not already open.
2. Navigate to and expand the package in which you want the cloud connector service to reside. Right-click the folder in which you want to create the service and select **New > Cloud Connector Service**.

   Designer displays the New Cloud Connector Service wizard.
3. On the **Cloud Connector Service** page of the wizard, in the **Element name** field, type the name you want to assign to the cloud connector service. Click **Next**.
4. On the **Connector** page of the wizard, select the CloudStreams Connector associated with the cloud application provider you want to access. Click **Next**.

   Tip:

If the list of available connectors is long and you know the name of the connector you want to use, you can locate the connector quickly by typing its name in the box below **Available Connectors**. You can also use this technique when selecting the connection pool and service in the next steps.

5. On the **Connection Pool** page of the wizard, select the connection pool for connecting to the cloud application provider. Click **Next**.

6. On the **Select Service** page of the wizard, select the cloud virtual service that you want the cloud connector service to invoke.

   Note:
   If only one cloud virtual service is available to select, this page will not appear.

7. Click **Finish**.

   Designer creates the cloud connector service and displays the service details in the cloud connector service editor.

8. Edit the cloud connector service as follows:

For this type of provider...	Follow the steps described in...
SOAP-based	"Editing a Cloud Connector Service for a SOAP-Based Provider" on page 389
REST-based	"Editing a Cloud Connector Service for a REST-Based Provider" on page 394

Editing a Cloud Connector Service for a SOAP-Based Provider

Editing a cloud connector service for a SOAP-based provider consists of specifying the operation, the business object associated with the operation, the headers to include in the service, the input/output signature that determines how the user interacts with the service, optional parameters to include in the input/output signature, and descriptive comments or usage notes, if any. You edit a cloud connector service using the service editor in Designer.

Keep the following points in mind when editing a cloud connector service:

- Before you edit a cloud connector service, create the service as described in "Creating a Cloud Connector Service" on page 388.
- Software AG CloudStreams provides a default connector virtual service for policy enforcements, called WmCloudStreams.SoapVS. If this service does not meet the needs of your CloudStreams project, ensure that an appropriate connector virtual service has been created for your project. For more information about CloudStreams connector virtual services, see *Administering webMethods CloudStreams*.

- In pipeline, document, and input/output validation, the data validation applies *constraints* to its variables. Constraints are the restrictions on the structure or content of variables. For more information about icons for constrained variables, see "Viewing the Constraints Applied to Variables" on page 401.

To edit a cloud connector service for a SOAP-based provider

1. Open Designer if it is not already open.

2. Navigate to and open the cloud connector service you created in "Creating a Cloud Connector Service" on page 388.

   The service opens in the cloud connector service editor.

3. On the **Operation** tab, from the **Connector Virtual Service** list, select the connector virtual service to be used for policy enforcement.

   For more information about CloudStreams connector virtual services, see *Administering webMethods CloudStreams*.

4. To configure the operation, business object, fields, and data types of fields, click next to **Operation**. Designer displays the **Operation and Business Object Configuration** wizard.

   a. Select the operation you want the cloud connector service to execute, and then click **Next**.

      When you change an operation, Designer clears all the metadata that were associated with the previously selected operation, including the headers, parameters, and data types of fields. You can select the metadata that the updated operation requires in the next steps.

      Note: Designer displays the appropriate pages of the **Operation and Business Object Configuration** wizard depending on whether the selected operation requires metadata, such as a business object, fields, and data types of fields. Further, Designer displays different **Business Object** panels based on the scenarios mentioned in the *Single or Multiple Operations and Multiple Business Objects with dependencies* section.

   b. In the **Select the Business Object** page, select a business object and click **Next**.

      Note:
      Designer displays nested, hierarchical, or multi-level business objects if the operation is designed to support nested business objects. You can expand the nested business objects to display the child-level objects.

   c. In the **Select Fields** page, specify the fields or parameters to use in the request/response body for the object.

      The mandatory fields or parameters for the business object are selected by default, and cannot be cleared.

d. You can add new custom fields as a String, String list, String Table, Document, Document list, Document reference, Document reference list, Object, and Object list by clicking and entering the custom field details in the **Add a new custom field** dialog box. You can add custom fields only for the operations that support custom fields. Custom field names should be unique within the available fields. While adding a custom field as the child of another custom field, the field name must be unique among all the children of the parent field. The following table lists the different toolbar buttons available in the **Select Fields** page:

Select...	To...
	Collapses all of the expanded fields.
	Add a new custom field.
	Edit a custom field.
	Delete a custom field.
	Move a custom field down in the list.
	Move a custom field up in the list.
	Promote a custom field in the hierarchy (that is, move the field one level up in the hierarchy).
	Demote a custom field in the hierarchy (that is, make the selected custom field a child of the preceding parent custom field).

e. If you want to configure concrete types for the abstract types in the operation you selected, click **Next**. If the operation you selected does not have any abstract type field, click **Finish**.

f. In the **Configure Data Types of Fields** page, select a value from the list of values next to the abstract type to configure concrete types for the abstract types in the operation.

g. Click **Finish**. Designer displays a confirmation message. Click **OK** to update the operation. Designer replaces the existing operation and associated metadata with the updated or default information.

Single or Multiple Operations with Multiple Business Objects with dependencies

Designer displays different **Business Object** panels based on the following scenarios:

Note:
Examples of **Interactions** (operations) are Create, Update, Upsert, and Delete. Examples of **Objects** are Contact, Account, and so on. **Requires** are dependencies, that is, a business object is dependent on another business object.

- **Single Operation has a single Business Object** - This panel appears if an operation has only a single business object. The operation has neither multiple interactions nor has records.

Only the object is displayed in the panel. An example of a single operation and a single object can be a "create" operation that contains only the "contact" business object.

- **Single Operation has multiple Business Objects** - This panel appears when a single operation has multiple business objects. The panel has an *optional* Record Number column (may not appear for some operations) and an Objects column. An example of a single operation with multiple objects can be a "create" operation that contains two business objects, "contact" and "account".
- **Single Operation has multiple Business Objects with dependencies** - This panel appears when a single operation has multiple business objects and some of the business objects may have dependencies on other business objects. The panel has Record Number, Objects, and the Requires columns. Here, you can set dependencies on the records appearing at a lower level, for example, record 3 can be dependent on (requires) record 1 or record 2.
- **Multiple Operations have multiple Business Objects** - This panel appears when multiple operations have multiple business objects. The panel has an *optional* Record Number column (may not appear for some operations), an Interactions column, and an Objects column. For example, the "create" and "update" operations can act on the "account" and "contact" business objects respectively.
- **Multiple Operations have multiple Business Objects with dependencies** - This panel appears when multiple operations have multiple business objects and some of the business objects may have dependencies on other business objects. The panel has Record Number, Interactions, Objects, and the Requires columns. For example, the "create" and "update" operations can act on the "account" and "contact" business objects respectively. Also, you can set dependencies on the records appearing at a lower level, for example, record 3 can be dependent on (requires) record 1 or record 2.

5. On the **Headers** tab, do the following:

   a. To include a header as part of the service signature, select the **Active** check box next to the header.

   b. To specify a default value for the header variable, click the **Default Value** box next to the variable and type or paste a default value. If the variable is null in the input pipeline, this default value will be used at run time. The value given at run time always take precedence over the default value. However, if the existing default value is of type fixed default, the overwrite will fail.

   c. Repeat the above steps in the **Output** section of the tab to select the SOAP headers whose contents you want to add to the service's output pipeline.

Note:
If the operation you selected on the **Operation** tab has mandatory headers, Designer displays those headers in gray. You cannot edit or delete a mandatory header.

6. If the operation you selected has predefined input parameters (for example, the Query and QueryAll operations have the `where` and `limit` parameters), you can configure them on the Parameters tab as follows:

   a. To specify a default value for a parameter, click the **Default Value** box next to the parameter. Then, type or paste a default value. If the variable is null in the input pipeline, this default value will be used at run time. The value given at run time always take precedence over the default value. However, if the existing default value is of type fixed default, the overwrite will fail.

   b. If a predefined parameter is not mandatory, you can activate/de-activate the parameter by clicking the **Active** check box.

      If a predefined parameter is mandatory, Designer displays the parameter in gray and the **Active** check box is selected. You cannot de-activate or delete a mandatory parameter.

   c. To move a parameter up in the list, select the parameter and click . To move a parameter down in the list, select the parameter and click .

7. If you want to add other parameters to the service signature, such as variables to be replaced at run time with a user's input, do the following on the **Parameters** tab:

   a. Click .

   b. Assign a name to the new parameter. If you want to rename the parameter later, click its name and type a new name.

   c. To specify a default value for the parameter, click the **Default Value** box next to the parameter. Then, type or paste a default value. If the variable is null in the input pipeline, this default value will be used at run time. The value given at run time always take precedence over the default value. However, if the existing default value is of type fixed default, the overwrite will fail.

   d. You can activate/de-activate the parameter by clicking the **Active** check box, or you can delete it by selecting the parameter and clicking .

   e. To move a parameter up in the list, select the parameter and click . To move a parameter down in the list, select the parameter and click .

8. On the **Input/Output** tab, do the following:

   a. To have the server validate the input to the service against the service input signature, select the **Validate input** check box.

b. To have the server validate the output to the service against the service output signature, select the **Validate output** check box.

c. Review the service's input and output signature and make any necessary changes as follows:

To change the...	Go to the...
Virtual service name, operation, business object, fields, and data types of fields	Operation tab
List of headers in the requestHeaders or responseHeaders sections, or their default values	Headers tab
List of parameters in the parameters section, or their default values	Parameters tab

The requestBody and responseBody sections are derived from the operation you selected on the Operation tab. The value of $connectionAlias is derived from the connection pool you specified when you first created the cloud connector service. The fault section is derived from the operation response. You cannot change these values in the editor.

9. On the **Logged Fields** tab, do the following:

   a. Select the check boxes next to the fields you want to log at run time.

   b. If you want to create an alias for a logged field to make it easier to locate in Designer, click the **Alias** box next to a field and type the alias name.

   For more information about logged fields, see the section on logging input and output fields in Designer.

10. On the **Summary** tab, review the details about the cloud connector service.

11. On the **Comments** tab, enter descriptive comments or usage notes, if any.

12. Click **File** > **Save** to save your changes.

Editing a Cloud Connector Service for a REST-Based Provider

Editing a cloud connector service for a REST-based provider consists of specifying the resource, the type of processing for requests or responses, the headers to include in the service, the input/output signature that determines how the user interacts with the service, default values for parameters included in the input/output signature, and descriptive comments or usage notes, if any. You edit a cloud connector service using the service editor in Designer.

Keep the following points in mind when editing a cloud connector service:

- Before you edit a cloud connector service, create the service as described in "Creating a Cloud Connector Service" on page 388.

- Software AG CloudStreams provides a default connector virtual service for policy enforcements, called WmCloudStreams.RestVS. If this service does not meet the needs of your CloudStreams project, ensure that an appropriate connector virtual service has been created for your project. For more information about CloudStreams connector virtual services, see Administering webMethods CloudStreams.
- In pipeline, document, and input/output validation, the data validation applies *constraints* to its variables. Constraints are the restrictions on the structure or content of variables. For more information about icons for constrained variables, see "Viewing the Constraints Applied to Variables" on page 401.

To edit a cloud connector service for a REST-based provider

1. Open Designer if it is not already open.
2. Navigate to and open the cloud connector service you created in "Creating a Cloud Connector Service" on page 388.

   The service opens in the cloud connector service editor.
3. On the **Resource** tab, do the following:

   a. From the **Connector Virtual Service** list, select the connector virtual service to be used for policy enforcements.

      For more information about CloudStreams connector virtual services, see *Administering webMethods CloudStreams*.

   b. Click next to **Resource Name**. Designer displays the **Resource and Business Object Configuration** wizard.

   c. Select the REST resource you want the cloud connector service to process, and then click **Next**.

      When you change a resource, Designer clears all the metadata that were associated with the previously selected resource, including the headers, parameters, and data types of fields. You can select the metadata that the updated resource requires in the next steps.

      Note:
      Designer displays the appropriate pages of the **Resource and Business Object Configuration** wizard depending on whether the selected resource requires metadata, such as a business object, fields, and data types of fields. Further, Designer displays different **Business Object** panels based on the scenarios mentioned in the *Single or Multiple Resources and Multiple Business Objects with dependencies* section.

   d. In the **Select the Business Object** page, select a business object and click **Next**.

      Note:

Designer displays nested, hierarchical, or multi-level business objects if the resource is designed to support nested business objects. You can expand the nested business objects to display the child-level objects.

e. In the **Select Fields** page, specify the fields or parameters to use in the request/response body for the object.

The mandatory fields or parameters for the business object are selected by default, and cannot be cleared.

f. You can add new custom fields as a String, String list, String Table, Document, Document list, Document reference, Document reference list, Object, and Object list by clicking and entering the custom field details in the **Add a new custom field** dialog box. You can add custom fields only for the operations that support custom fields. Custom field names should be unique within the available fields. While adding a custom field as the child of another custom field, the field name must be unique among all the children of the parent field. The following table lists the different toolbar buttons available in the **Select Fields** page:

Select...	To...
	Collapses all of the expanded fields.
	Add a new custom field.
	Edit a custom field.
	Delete a custom field.
	Move a custom field down in the list.
	Move a custom field up in the list.
	Promote a custom field in the hierarchy (that is, move the field one level up in the hierarchy).
	Demote a custom field in the hierarchy (that is, make the selected custom field a child of the preceding parent custom field).

g. If you want to configure concrete types for the abstract types in the resource you selected, click **Next**. If the resource you selected does not have any abstract type field, click **Finish**.

h. In the **Configure Data Types of Fields** page, select a value from the list of values next to the abstract type to configure concrete types for the abstract types in the resource.

i. Click **Finish**. Designer displays a confirmation message. Click **OK** to update the resource. Designer replaces the existing resource and associated metadata with the updated or default information.

j. In the **Request Processing** section, select an appropriate parsing type. The parsing type determines how the service accepts the input.

Option	Meaning
Document	Builds the request message as an IS document type. Select this option when the provider's XML file includes a schema or specification describing the content of the request.
Binary Stream	Builds the request message as a binary stream. Select this option when you expect the pipeline to contain an input stream for which no document type exists or when it is not practical to provide a schema description of the content. For example, the content that is posted for the Salesforce.com "createBatch" resource has a complex structure of fields and rows. A batch of new accounts can be created, and each account can have dozens of fields with precise formatting requirements (for example, date fields). Attachments can even be included in the batch file. The stream option is the best option for this type of resource.

Note:
If the resource you selected does not contain any requests or responses, the Request Processing or Response Processing fields are not available.

k. In the **Response Processing** section, select an appropriate serialization type. The serialization type constructs the cloud connector service's output signature and determines how the cloud connector service should return data to the user.

Option	Meaning
Document	Formats the response message as an IS document type. Select this option when the provider's XML file includes a schema or specification describing the content of the response.
Binary Stream	Formats the response message as a binary stream. Select this option when you expect the pipeline to contain an output stream for which no document type exists or when it is not practical to provide a schema description of the content. **Note:** This option works in conjunction with the response's parsing type property. If you select **Stream** as the response's serialization type, Designer also selects **Stream** as the response's parsing type.

Note:
If the resource you selected does not contain any requests or responses, the Request Processing or Response Processing fields will not be available.

Single or Multiple Resources with Multiple Business Objects with dependencies

Designer displays different **Business Object** panels based on the following scenarios:

Note:
Examples of **Interactions** (resources) are Create, Update, Upsert, and Delete. Examples of **Objects** are Contact, Account, and so on. **Requires** are dependencies, that is, a business object is dependent on another business object.

- **Single Resource has a single Business Object** - This panel appears if a resource has only a single business object. The resource has neither multiple interactions nor has records. Only the object is displayed in the panel. An example of a single resource and a single object can be a "create" resource that contains only the "contact" business object.
- **Single Resource has multiple Business Objects** - This panel appears when a single resource has multiple business objects. The panel has an *optional* Record Number column (may not appear for some resources) and an Objects column. An example of a single resource with multiple business objects can be a "create" resource that contains two business objects, "contact" and "account".
- **Single Resource has multiple Business Objects with dependencies** - This panel appears when a single resource has multiple business objects and some of the business objects may have dependencies on other business objects. The panel has Record Number, Objects, and the Requires columns. Here, you can set dependencies on the records appearing at a lower level, for example, record 3 can be dependent on (requires) record 1 or record 2.
- **Multiple Resources have multiple Business Objects** - This panel appears when multiple resources have multiple business objects. The panel has an *optional* Record Number column (may not appear for some resources), an Interactions column, and an Objects column. For example, the "create" and "update" resources can act on the "account" and "contact" business objects respectively.
- **Multiple Resources have multiple Business Objects with dependencies** - This panel appears when multiple resources have multiple business objects and some of the business objects may have dependencies on other business objects. The panel has the Record Number, Interactions, Objects, and Requires columns. For example, the "create" and "update" resources can act on the "account" and "contact" business objects respectively. Also, you can set dependencies on the records appearing at a lower level, for example, record 3 can be dependent on (requires) record 1 or record 2.

4. On the **Headers** tab, Designer displays the default HTTP transport headers for the resource, along with their default values. At run time, while processing the headers, Software AG CloudStreams substitutes values as necessary, for example, replaces the “cn.sessionToken” value in the X-SFDC-Session header with the actual runtime session ID. In order to customize the headers, do the following:

a. To specify a default value for the header variable, click the **Default Value** box to the right of the variable and type or paste the new value. If the variable is null in the input pipeline, this value will be used at run time. If the variable has an existing default value defined in the Cloud Connector Descriptor, this value will overwrite the existing value at run time. However, if the existing default value is of type "fixed default", the overwrite will fail as mentioned earlier.

b. To add a custom header to the service's input pipeline, in the Input section of the tab, click . Type a name for the header and provide a default value if desired.

c. To move a header up in the list, select the header and click . To move a header down in the list, select the header and click .

d. To include a header as part of the service signature, select the **Active** check box next to the header.

e. To delete a custom header that you added, select the header and click .

> **Note:**
> You cannot delete the resource's required headers.

f. Repeat the above steps in the Output section of the tab to select the HTTP transport protocol headers whose contents you want to add to the service's output pipeline.

> **Note:**
> A provider's response headers only appear in the pipeline signature if they are added as active output headers in the Output section. Any unspecified headers returned by the native provider will not be included in the pipeline.

5. On the **Parameters** tab, Designer displays the configured resource parameters. In order to customize the parameters, do the following:

a. Review the details about the resource parameters. Designer displays the parameter name and description, the data type used to represent the kind of information the parameter can hold, the parameterization style of the request, and the dynamic default value needed to access the resource.

Currently, three parameter styles are supported: *URI_CONTEXT*, *QUERYSTRING_PARAM* and *CFG_PARAM*.

For more information about the supported parameter styles, see the section *Understanding REST Parameters* in the document *Administering webMethods CloudStreams*.

b. To specify a default value for the parameter, click the **Default Value** box to the right of the parameter. Then, type or paste the default value. The default value is used at run time, if the parameter value is not explicitly specified in the input pipeline. Also, this default value will overwrite any existing default value that is defined in the Cloud Connector Descriptor,

at run time. However, if the existing default value is of type “fixed default”, the overwrite will fail as mentioned earlier.

Note:
You cannot specify a default value for a parameter with data type as “Record”.

6. On the **Input/Output** tab, do the following:

   a. To have the server validate the input to the service against the service input signature, select the **Validate input** check box.

   b. To have the server validate the output to the service against the service output signature, select the **Validate output** check box.

   c. Review the service’s input and output signature and make any necessary changes as follows:

To change the...	Go to the...
List of headers in the requestHeaders or responseHeaders section, or their default values	Headers tab
Default value of a parameter in the parameters section, or their default values	Parameters tab

   The requestBody and responseBody sections are derived from the REST resource you selected on the Resource tab. The value of $connectionAlias is derived from the connection pool you specified when you first created the cloud connector service. The status, statusMessage, and fault values are derived from the resource response. You cannot change these values in the editor.

7. On the **Logged Fields** tab, do the following:

   a. Select the check boxes next to the fields you want to log at run time.

   b. If you want to create an alias for a logged field to make it easier to locate in Designer, click the **Alias** box next to a field and type the alias name.

   For more information about logged fields, see the section on logging input and output fields in Designer.

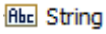
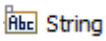
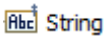
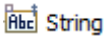
8. On the **Summary** tab, review the details about the cloud connector service.

9. On the **Comments** tab, enter descriptive comments or usage notes, if any.

10. Click **File > Save** to save your changes.

Viewing the Constraints Applied to Variables

Designer displays small symbols next to a variable icon to indicate the constraints applied to the variable. Designer displays variables in the following ways:

Variable	Constraint Status	Variable Properties
	Required field.	The **Required** property is set to True.
	Optional field.	The **Required** property is set to False.
	Required field with content type constraint.	The **Content type** property specifies an IS schema or XML schema.
	Optional field with content type constraint.	The **Required** property is set to False, and the Content type property specifies an IS schema or XML schema.
	Required field with default value.	The **Fixed** property is set to False, and the defaultValue property specifies a default value. The variable has a default value, but you can override this default value with any other valid values while executing the service or mapping the variables.
	Required field with fixed value.	The **Fixed** property is set to True, and the **defaultValue** property .specifies a null value. The variable has a null value assigned to it by default and you cannot override this value. You cannot map this variable to another variable or assign any input values to this variable during service execution.
	Required field with fixed default value.	The **Fixed** property is set to True, and the **defaultValue** property specifies a default value. The variable has a default value and you cannot override this value. You cannot map this variable to another variable or assign any input values to this variable during service execution.

21 Running Services

- Using Launch Configurations to Run Services 404
- Supplying Input Values to a Service 406
- Running a Service 421
- Viewing Results from Running a Service 422
- Running Services from a Browser 426

When you run a service, Designer invokes the service (just as an ordinary IS client would) and receives its results. The service executes once, from beginning to end (or until an error condition forces it to stop). The service executes on the Integration Server on which you have an open session, or if you using a launch configuration, on the Integration Server specified in the launch configuration.

Results from the service are returned to Designer and displayed in Results view. This allows you to quickly examine the data that the service produces and optionally change it or save it to a file. You can use the saved data as input for a later debug session or to populate the pipeline during a debugging session.

Using Launch Configurations to Run Services

In Designer, you create launch configurations to run services. A launch configuration contains the information Designer needs to execute a service. You can create one or more launch configurations for each service.

When running the service, Designer invokes the service (just as an ordinary IS client would) and receives its results. The service executes once, from beginning to end (or until an error condition forces it to stop) on the Integration Server on which the service resides.

Note:
You also create launch configurations to debug flow services. You can use a launch configuration created for running a service when you debug a flow service. Similarly, you can use a launch configuration that you created for debugging a flow service when you run a service. For more information about launch configurations for debugging flow services, see "Creating Launch Configurations for Debugging Flow Services" on page 432.

Designer requires launch configurations to run services. However, if a service does not have an associated launch configuration and you bypass the Run Configurations dialog boxes when running the service, Designer creates one on the fly and saves it in your workspace. You can use this configuration from one session to the next. In fact, Designer reuses this configuration every time you run or debug the service without creating another launch configuration.

By default, Designer saves launch configurations locally in an unexposed location in your workspace. However, you might want to share launch configurations with other developers. You can specify that Designer save a launch configuration to a shared file within your workspace; this location will be exposed. On the **Common** tab in the Run Configurations dialog box, select the **Shared file** option and provide a workspace location in which to save the file.

You might consider creating a launch configuration for each set of data that you routinely use to test your service. This will provide you with a ready-made set of test cases against which to verify the service when it is modified by you or other developers in the future. Many sites establish a workspace project directory just for holding sets of test data that they generate in this manner.

Creating a Launch Configuration for Running a Service

Use the following procedure to create a launch configuration for use in running a service.

To create a launch configuration for running a service

1. In Designer: **Run > Run Configurations**

2. In the **Run Configuration** dialog box, select **IS Service** and click to add a new launch configuration.

3. In the **Name** field, specify a name for the launch configuration

4. On the **Service** tab, in the **Integration Server** list, select the Integration Server on which the service for which you are creating a launch configuration resides.

5. In the **Service** field, enter the name of the service for which you are creating a launch configuration of click **Browse** to select the service.

6. Specify input values to pass the service.

   a. On the **Input** tab, select **Use IData**.

   b. Specify the input values to save with the launch configuration by doing one of the following:

      - Type the input value for each service input parameter. For more information about providing input values, see "Entering Input for a Service" on page 406.
      - To load the input values that match the structure of the service input signature from a file, click **Load** to locate and select the file containing the input values.
      - To load input values from a file and replace the service input signature with the structure and data types in the file, click **Load and Replace**.

      If Designer cannot parse the input data, it displays an error message indicating why it cannot load the data. For more information about loading input values from a file, see "Loading Input Values" on page 420.

   c. If you want to pass empty variables (variables that have no value) to the service, select the **Include empty values for String Type** check box. When you select this option, empty strings are passed with a zero-length value. If you do not select this option, Designer excludes empty value from the IData it passes to the service as input.

7. If you want to save the input values that you have entered, click **Save**. Input values that you save can be recalled and reused in later tests.

8. Click **Apply**.

9. If you want to execute the launch configuration, click **Run**. Otherwise, click **Close**.

Supplying Input Values to a Service

When you create a launch configuration, run a service, or debug a service you need to pass input to the service. You can provide a value for every input parameter in the service signature. You can save a set of input values to a file and reuse them in later tests or with other launch configurations. When debugging a flow service, as an alternative you can pass the service an XML document as input.

Entering Input for a Service

When you create a launch configuration for a service, run a service, or debug a service Designer displays a screen for specifying input values for the variables in the service's input signature. Designer validates the input values you specify. If specified values do not match the input parameter data type, Designer displays a message to that effect.

Although you can specify input values for most input variables, you *cannot* specify input values for:

- Document List variables that have no defined content
- Object variables constrained as a byte [ ]
- Unconstrained Objects (Objects of unknown type)

If the inputs to the service contain one of these variables for which you cannot specify input, to test the service create another service that generates the input values for the service. Then construct a test harness, which is a flow service that executes both the service that generates the test input values and the service you want to test. Finally, to test the service, execute the test harness.

To enter input values for a service

1. Do one of the following to display a window for providing inputs:
   - Open an IS Service launch configuration to run a service as described in "Creating a Launch Configuration for Running a Service" on page 404.
   - Run a service as described in "Running a Service" on page 421.
   - Open an IS Service launch configuration to debug a flow service as described in "Creating Launch Configurations for Debugging Flow Services" on page 432.
   - Launch a Java class in debug mode as described in "Debugging a Java Service while its Class Runs in Designer " on page 462.
2. If you are working with a launch configuration, select the **Input** tab to display the screen you use to specify input.

> **Note:**
> If you are running or debugging the service, Designer displays the **Enter Input for *serviceName*** dialog box that you use to specify input.

3. To load input values from a file, do one of the following:

   - If you want to use inputs that match the structure of the service input signature, on the **Input** tab click **Load**.
   - If you want the data structure of the input values in a file to overwrite the structure of the service input signature, click **Load and Replace**.

4. Select or clear the **Include empty values for String Types** check box to indicate how to handle variables that have no value.

   - If you want to use an empty String (i.e., a String with a zero-length), select the **Include empty values for String Types** check box. Also note that Document Lists that have defined elements will be part of the input, but they will be empty.
   - If you want to use a null value for the empty Strings, clear the check box. String-type variables will not included in the input document.

   Note:
   The setting applies to all String-type variables in the root document of the input signature. The setting does not apply to String-type variables within Document Lists. You define how you want to handle String-type variables within Document Lists separately when you assign values to Document Lists variables. For more information, see "Specifying Values for a Document List Variable" on page 416.

5. Enter values for the input variables. For specific information for how to specify a value based on a variable's data type, see one of the following:

For this type of variable...	See...
String	"Specifying a Value for a String Variable" on page 408
String List	"Specifying Values for a String List Variable" on page 409
String Table	"Specifying Values for a String Table Variable" on page 411
Document Document Reference	"Specifying Values for a Document Variable that Has Defined Content" on page 413 or "Specifying Values for a Document Variable with No Defined Content" on page 414
Document List Document Reference List	"Specifying Values for a Document List Variable" on page 416
Object	"Specifying a Value for an Object Variable" on page 418
Object List	"Specifying Values for an Object List Variable" on page 419

6. To save the input values to a file for use in later debugging, click **Save** or **Save Inputs**. In the **Save As** dialog box, specify the name and location of the file to which you want the values saved. Click **Save**.

Specifying a Value for a String Variable

Use the following procedure to specify a value for a String variable. You perform this procedure from:

- The **Input** tab if you are working with a launch configuration
- The **Enter Input for *serviceName*** dialog box if you are running or debugging a service
- The **Enter Input for *variableName*** dialog box if you are assigning a value, pipeline variable, or global variable to a variable in the pipeline.

To specify a value for a String variable

1. In the **Value** column for the String variable:
   - If you want to assign a literal value to the variable, type a String value in the **Value** column.
   - If you want to derive the value from a String variable in the pipeline, type the name of that variable enclosed in % symbols (for example, `%Phone%`).
   - If you want to derive the value from a global variable, type the name of that global variable enclosed in % symbols (for example, `%myFTPServer%`).
   - You can mix literal values and variable names. For example, if you specify `(%areaCode%) %Phone%`, the resulting String would be formatted to include the parentheses and space. If you specify `%firstName% %initial%. %lastName%`, the period and spacing would be included in the value.
2. If you specified a pipeline variable as the value of the String variable (for example, `%Phone%`), select the **Perform pipeline variable substitution** check box so that during service execution Integration Server replaces the pipeline variable name with the run-time value of the variable.

   Note:
   The **Perform pipeline variable substitution** check box is not available when using a launch configuration.
3. If you specified a global variable as the value of the String variable (for example, `%myFTPUserName%`), select the **Perform global variable substitution** check box so that Integration Server replaces the variable name with the global variable value at run time.

   Note:
   The **Perform global variable substitution** check box is not available when using a launch configuration.

Note:
If a pipeline variable and a global variable have the same name and you select both the **Perform global variable substitution** and **Perform pipeline variable substitution** check boxes, Integration Server uses the value of the pipeline variable.

4. If you want Integration Server to use the value you specified only when the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server will always apply the value you specified.)

Note:
The **Overwrite pipeline value** check box is not available when using a launch configuration.

5. Do one of the following:

   - If you are working with a launch configuration, click **Apply** on the **Input** tab to save the value you entered. You can continue to specify values or click **Run** to execute the service.
   - If using the **Enter Input for *serviceName*** dialog box, continue to specify input values, or if you are finished, click **OK** to close the dialog box and execute the service.
   - If using the **Enter Input for *variableName*** dialog box, click **OK** to close the dialog box.

Specifying Values for a String List Variable

Use the following procedure to specify values for a String List variable. You perform this procedure from:

- The **Input** tab if you are working with a launch configuration
- The **Enter Input for *serviceName*** dialog box if you are running or debugging a service
- The **Enter Input for *variableName*** dialog box if you are assigning a value, pipeline variable, or global variable to a variable in the pipeline.

To specify values for a String List variable

1. Select the String List variable.

2. Do the following to append, insert, and delete Strings within the String List:

To...	Do this...
Append a String to the end of the list	Click **Add Row** and specify a value in the **Value** column.
Insert a String into the middle of the list	Select the String below where you want to add the new one and click **Insert Row**.
Remove a String from the list	Select the String and click **Delete Row**.

When specifying a value in the **Value** column:

- If you want to assign a literal value to the variable, type a String value in the **Value** column.
- If you want to derive the value from a String variable in the pipeline, type the name of that variable enclosed in % symbols (for example, `%Phone%`).
- If you want to derive the value from a global variable, type the name of that global variable enclosed in % symbols (for example, `%myFTPServer%`).
- You can mix literal and pipeline variables. For example, if you specify `(%areaCode%) %Phone%`, the resulting String would be formatted to include the parentheses and space. If you specify `%firstName% %initial%. %lastName%`, the period and spacing would be included in the value.

3. If you assigned a value using the % symbol along with a pipeline variable (for example, `%Phone%`), select the **Perform pipeline variable substitution** check box so that during service execution Integration Server replaces the pipeline variable name with the run-time value of the variable.

   Note:
   The **Perform pipeline variable substitution** check box is not available when using a launch configuration.

4. If you specified a global variable as the value of the String variable (for example, `%myFTPUserName%`), select the **Perform global variable substitution** check box so that Integration Server replaces the global variable name with the global variable value at run time.

   Note:
   The **Perform global variable substitution** check box is not available when using a launch configuration.

   Note:
   If a pipeline variable and a global variable have the same name and you select both the **Perform global variable substitution** and **Perform pipeline variable substitution** check boxes, Integration Server uses the value of the pipeline variable.

5. If you want Integration Server to use the value you specified only when the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server always applies the value you specified.)

   Note:
   The **Overwrite pipeline value** check box is not available when using a launch configuration.

6. After adding the String List elements you want and specifying values, do one of the following:

   - If you are working with a launch configuration, click **Apply** on the **Input** tab to save the value you entered. You can continue to specify values or click **Run** to execute the service.

- If using the **Enter Input for *serviceName*** dialog box, continue to specify input values, or if you are finished, click **OK** to close the dialog box and execute the service.
- If using the **Enter Input for *variableName*** dialog box, click **OK** to close the dialog box.

Specifying Values for a String Table Variable

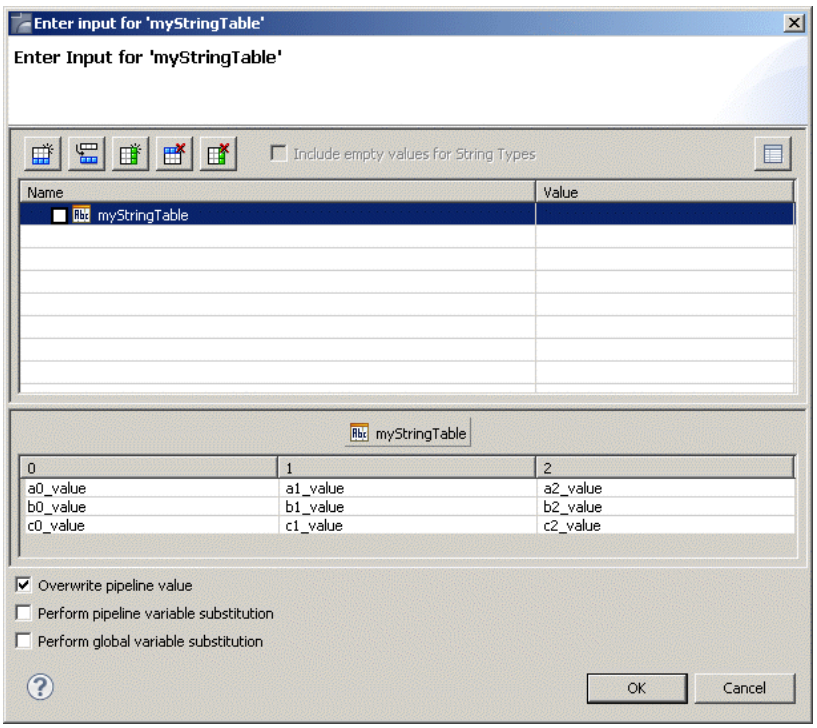
Use the following procedure to specify values for a String Table variable. You perform this procedure from:

- The **Input** tab if you are working with a launch configuration
- The **Enter Input for *serviceName*** dialog box if you are running or debugging a service
- The **Enter Input for *variableName*** dialog box if you are assigning a value, pipeline variable, or global variable to a variable in the pipeline.

To specify values for a String Table variable

1. Select the String Table variable.

   Designer displays a table for the String Table variable at the bottom of the dialog box. Use the table to add rows and columns and assign values.



2. Do the following to add/insert/delete rows and columns in the table:

To...	Do this...
Append a row to the end of the table	Click **Add Row**.
Insert a row in the middle	In the table viewer, select the row below where you want to add the new one and click **Insert Row**.
Remove a row	Select the row in the table viewer and click **Delete Row**.
Add a column to the table	Click **Add Column**.
Remove a column from the table	Click **Delete Column**.

3. Type values in the cells of the table.

   - If you want to assign a literal value, type a String value.
   - If you want to derive the value from a String variable in the pipeline, type the name of that variable enclosed in % symbols (for example, `%Phone%`).
   - If you want to derive the value from a global variable, type the name of that global variable enclosed in % symbols (for example, `%myFTPUserName%`.).
   - You can mix literal and pipeline variables. For example, if you specify `(%areaCode%) %Phone%`, the resulting String would be formatted to include the parentheses and space. If you specify `%firstName% %initial%. %lastName%`, the period and spacing would be included in the value.

4. If you assigned a value using the % symbol along with a pipeline variable (for example, `%myFTPServer%`), select the **Perform pipeline variable substitution** check box so that during service execution Integration Server replaces the pipeline variable name with the run-time value of the variable.

   Note:
   The **Perform pipeline variable substitution** check box is not available when using a launch configuration.

5. If you specified a global variable as the value of the String variable (for example, `%myFTPUserName%`), select the **Perform global variable substitution** check box so that Integration Server replaces the variable name with the global variable value at run time.

   Note:
   The **Perform global variable substitution** check box is not available when using a launch configuration.

   Note:

If a pipeline variable and a global variable have the same name and you select both the **Perform global variable substitution** and **Perform pipeline variable substitution** check boxes, Integration Server uses the value of the pipeline variable.

6. If you want Integration Server to use the value you specified only when the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server will always apply the value you specified.)

Note:
The **Overwrite pipeline value** check box is not available when using a launch configuration.

7. After adding the table rows and columns you want and assigning values, do one of the following:

- If you are working with a launch configuration, click **Apply** on the **Input** tab to save the value you entered. You can continue to specify values or click **Run** to execute the service.
- If using the **Enter Input for *serviceName*** dialog box, continue to specify input values, or if you are finished, click **OK** to close the dialog box and execute the service.
- If using the **Enter Input for *variableName*** dialog box, click **OK** to close the dialog box.

Specifying Values for a Document Variable that Has Defined Content

You can assign values to Document variables that have defined content, including referenced documents and recursive documents. If part of the document tree is collapsed, you can expand it.

Tip:
Click to view and/or update your preferences for how Designer displays and expands the contents of Document variables.

Note:
If the Document variable has no defined content, you can add String name/value pairs and then assign values. For more information, see "Specifying Values for a Document Variable with No Defined Content" on page 414.

Use the following procedure to specify values for a Document variable. You perform this procedure from:

- The **Input** tab if you are working with a launch configuration
- The **Enter Input for *serviceName*** dialog box if you are running or debugging a service
- The **Enter Input for *variableName*** dialog box if you are assigning a value to a pipeline variable

To specify values for a Document variable that has defined content

1. Select the variables within the Document, and provide values. For help for how to specify a value based on the variable's data type, see one of the following:

For this type of variable...	See...
String	"Specifying a Value for a String Variable" on page 408
String List	"Specifying Values for a String List Variable" on page 409
String Table	"Specifying Values for a String Table Variable" on page 411
Document Document Reference	"Specifying Values for a Document Variable that Has Defined Content" on page 413 or "Specifying Values for a Document Variable with No Defined Content" on page 414
Document List Document Reference List	"Specifying Values for a Document List Variable" on page 416
Object	"Specifying a Value for an Object Variable" on page 418
Object List	"Specifying Values for an Object List Variable" on page 419

2. If you want Integration Server to use the value you specified only when the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server will always apply the value you specified.)

   Note:
   The **Overwrite pipeline value** check box is not available when using a launch configuration.

3. After assigning values, do one of the following:

   - If you are working with a launch configuration, click **Apply** on the **Input** tab to save the value you entered. You can continue to specify values or click **Run** to execute the service.
   - If using the **Enter Input for *serviceName*** dialog box, continue to specify input values, or if you are finished, click **OK** to close the dialog box and execute the service.
   - If using the **Enter Input for *variableName*** dialog box, click **OK** to close the dialog box.

Specifying Values for a Document Variable with No Defined Content

If the Document variable has no defined content, you can add String name/value pairs and then assign values.

Note:
If the Document already has defined content, see "Specifying Values for a Document Variable that Has Defined Content" on page 413.

Use the following procedure to specify values for a Document variable. You perform this procedure from:

- The **Input** tab if you are working with a launch configuration
- The **Enter Input for *serviceName*** dialog box if you are running or debugging a service
- The **Enter Input for *variableName*** dialog box if you are assigning a value to a pipeline variable

To specify String name/value pairs for a Document variable with no defined content

1. Select the Document variable that has no defined content.

   Designer displays a document viewer at the bottom of the screen. You use the document viewer to add String name/value pairs.

2. Do the following to append, insert, and delete String name/value pairs:

To...	Do this...
Append a name/value pair to the end	Click **Add Row**.
Insert a name/value pair in the middle	In the document viewer, select the name/value pair below where you want to add the new one and click **Insert Row**
Remove a name/value pair	Select the name/value pair in the document viewer and click **Delete Row**

3. For each name/value pair you added, in the **Name** column type a name.

   Note:
   If you leave a **Name** column empty, Designer will discard the row.

4. For each name/value pair you added, in the **Value** column type a value:

   - If you want to assign a literal value, type a String value.
   - If you want to derive the value from a String variable in the pipeline, type the name of that variable enclosed in % symbols (for example, `%Phone%`).
   - You can mix literal and pipeline variables. For example, if you specify `(%areaCode%) %Phone%`, the resulting String would be formatted to include the parentheses and space. If you specify `%firstName% %initial%. %lastName%`, the period and spacing would be included in the value.

5. If you assigned a value using the % symbol along with a pipeline variable (for example, `%Phone%`), select the **Perform pipeline variable substitution** check box so that the server performs variable substitution at run time.

Note:
The **Perform pipeline variable substitution** check box is not available when using a launch configuration.

6. If you want Integration Server to use the value you specified only when the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server will always apply the value you specified.)

Note:
The **Overwrite pipeline value** check box is not available when using a launch configuration.

7. After assigning values, do one of the following:

- If you are working with a launch configuration, click **Apply** on the **Input** tab to save the value you entered. You can continue to specify values or click **Run** to execute the service.
- If using the **Enter Input for *serviceName*** dialog box, continue to specify input values, or if you are finished, click **OK** to close the dialog box and execute the service.
- If using the **Enter Input for *variableName*** dialog box, click **OK** to close the dialog box.

Specifying Values for a Document List Variable

You can assign values to Document List variables that have defined content, including referenced document lists. When setting values, if part of a document tree is collapsed, you can expand.

Tip:
Click to view and/or update your preferences for how Designer displays and expands the contents of Document variables.

Use the following procedure to specify values for a Document List variable. You perform this procedure from:

- The **Input** tab if you are working with a launch configuration
- The **Enter Input for *serviceName*** dialog box if you are running or debugging a service
- The **Enter Input for *variableName*** dialog box if you are assigning a value to a pipeline variable

To specify values for a Document List variable

1. Select the Document List variable.

2. Do the following to append, insert, and delete Documents within the Document List:

To...	Do this...
Append a Document to the end of the list	Click **Add Row**.

To...	Do this...
Insert a Document into the middle of the list	Select the Document below where you want to add the new one and click **Insert Row**
Remove a Document from the list	Select the Document and click **Delete Row**

3. Assign values to the Document variables you added to the list.

   How you assign values is based on the data type of a variable. For help for how to specify a value for the variable, see one of the following:

For this type of variable...	See...
String	"Specifying a Value for a String Variable" on page 408
String List	"Specifying Values for a String List Variable" on page 409
String Table	"Specifying Values for a String Table Variable" on page 411
Document Document Reference	"Specifying Values for a Document Variable that Has Defined Content" on page 413 or "Specifying Values for a Document Variable with No Defined Content" on page 414
Document List Document Reference List	"Specifying Values for a Document List Variable" on page 416
Object	"Specifying a Value for an Object Variable" on page 418
Object List	"Specifying Values for an Object List Variable" on page 419

4. For each top-level Document that you added to the Document List, select how you want to handle String variables within that Document that have no value.

   - If you want to use empty Strings (i.e., a String with a zero-length), select either the check box next to the top-level Document in the tree or the **Include empty values for String Types** check box. Designer will select both check boxes.
   - If you want to use null values, clear the check box next to the top-level Document in the tree or the **Include empty values for String Types** check box. Designer will clear both check boxes.

   Designer allows you to define this setting for each Document within a Document List.

5. If you want Integration Server to use the value you specified only when the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server will always apply the value you specified.)

   Note:
   The **Overwrite pipeline value** check box is not available when using a launch configuration.

6. After assigning values, do one of the following:

   - If you are working with a launch configuration, click **Apply** on the **Input** tab to save the value you entered. You can continue to specify values or click **Run** to execute the service.
   - If using the **Enter Input for *serviceName*** dialog box, continue to specify input values, or if you are finished, click **OK** to close the dialog box and execute the service.
   - If using the **Enter Input for *variableName*** dialog box, click **OK** to close the dialog box.

Specifying a Value for an Object Variable

You can set values for most constrained Objects. You *cannot* assign values to:

- Objects constrained as a byte [ ]
- Unconstrained Objects (Objects of unknown type)

When you set values for constrained Objects, Designer automatically validates the values. If the value is not of the type specified by the Object constraint, Designer displays a message identifying the variable and the expected type.

Use the following procedure to specify value for an Object variable. You perform this procedure from:

- The **Input** tab if you are working with a launch configuration
- The **Enter Input for *serviceName*** dialog box if you are running or debugging a service
- The **Enter Input for *variableName*** dialog box if you are assigning a value to a pipeline variable

To specify a value for an Object variable

1. In the **Value** column, specify a value that is of the type defined by the Object constraint.

2. If you want Integration Server to use the value you specified only when the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server will always apply the value you specified.)

   Note:
   The **Overwrite pipeline value** check box is not available when using a launch configuration.

3. Do one of the following:

- If you are working with a launch configuration, click **Apply** on the **Input** tab to save the value you entered. You can continue to specify values or click **Run** to execute the service.
- If using the **Enter Input for *serviceName*** dialog box, continue to specify input values, or if you are finished, click **OK** to close the dialog box and execute the service.
- If using the **Enter Input for *variableName*** dialog box, click **OK** to close the dialog box

Specifying Values for an Object List Variable

You can set values for most constrained Object Lists. You *cannot* assign values to:

- Object Lists constrained as a byte [ ]
- Unconstrained Object Lists (Object Lists of unknown type)

When you set values for constrained Object Lists, Designer automatically validates the values. If the value is not of the type specified by the Object List constraint, Designer displays a message identifying the variable and the expected type.

Use the following procedure to specify values for an Object List variable. You perform this procedure from:

- The **Input** tab if you are working with a launch configuration
- The **Enter Input for *serviceName*** dialog box if you are running or debugging a service
- The **Enter Input for *variableName*** dialog box if you are assigning a value to a pipeline variable

To specify values for an Object List variable

1. Select the Object List variable.
2. Do the following to append, insert, and delete Objects within the Object List:

To...	Do this...
Append an Object to the end of the list	Click **Add Row**.
Insert an Object into the middle of the list	Select the Object below where you want to add the new one and click **Insert Row**
Remove an Object from the list	Select the Object and click **Delete Row**

3. If you want Integration Server to use the value you specified only when the variable does not contain a value at run time, clear the **Overwrite pipeline value** check box. (If you select this check box, Integration Server will always apply the value you specified.)

Note:
The **Overwrite pipeline value** check box is not available when using a launch configuration.

4. After adding the Object List elements you want and specifying values, do one of the following:

   - If you are working with a launch configuration, click **Apply** on the **Input** tab to save the value you entered. You can continue to specify values or click **Run** to execute the service.
   - If using the **Enter Input for *serviceName*** dialog box, continue to specify input values, or if you are finished, click **OK** to close the dialog box and execute the service.
   - If using the **Enter Input for *variableName*** dialog box, click **OK** to close the dialog box.

Saving Input Values

You can save the input values that you provide when creating a launch configuration, debugging a service, or when running a service, document type, or webMethods Messaging Trigger to a file. This enables you to reuse them in later tests or in other launch configurations.

When saving input values to a file, keep the following points in mind:

- Empty variables (variables that do not have a value) are only saved if the **Include empty values for String Types** check box is selected. If you do not select this check box, Designer does not save empty variables in the file.
- You can store the file in any directory that is accessible to the computer on which Designer is running. Because these files are not actual run-time components, they do not need to be saved in Integration Server's namespace or even on the server machine itself.
- Designer saves the data in XML format

Loading Input Values

When creating a launch configuration, debugging a service, or running a service, webMethods Messaging Trigger, or document type; you can reload input values that you have saved to a file instead of entering the values each time.

When you load input values from a file using the **Load** command, keep the following points in mind:

- Designer only loads variables whose name and type match those displayed in the Input dialog box. Variables that exist in the file, but not in the dialog box, are ignored. In the case of Object variables without constraints or Object variables of type byte[], the values in the file are not used.
- Values from the file replace those already in the Input tab or **Enter Input** dialog box.
- Variables that exist in the Input tab or **Enter Input** dialog box, but not in the file, are set to null.
- For flow services, besides loading values saved via the launch configuration or **Enter Input** dialog box, you can also load values that were saved using pub.flow:savePipelineToFile service. In addition, you can change values in the pipeline during debugging.

When you load input values from a file using the **Load and Replace** command, Designer overwrites the structure and data types of the input signature with the structure and data types defined in the file. This might be useful when the service input signature includes a document with an undefined structure. You can use **Load and Replace** to pass in documents with various content and content structure and therefore test or run the service with a variety of input data.

Running a Service

When you run a service, you can select the launch configuration that Designer uses to run the service. If a launch configuration does not exist for a service, Designer creates a launch configuration and immediately prompts you for input values and then runs the service. Designer saves the launch configuration in your workspace.

Note:
If a flow service expects an XML document as input, you must create a launch configuration and debug the service.

To run a service

1. In Package Navigator view, select the service you want to run.

2. In Designer: **Run > Run As > Run Service**

3. If multiple launch configurations exist for the service, use the **Select Launch Configuration** dialog box to select the launch configuration that you want Designer to use to run the service.

4. If the launch configuration is set up to prompt the user for input values or there is no launch configuration, in the **Enter Input for *serviceName*** dialog box, specify input values for the service.

   If the service has no input parameters, Designer displays the No input dialog box stating a message to that effect. If you do not want Designer to display this message when the service is run again, select the **Do not show this dialog again** check box. You can reverse this selection by selecting **Always show the No input dialog** on the Run/Debug preferences page.

5. Click **OK**. For more information about supplying input values, see "Entering Input for a Service" on page 406.

   Note:
   If you type in input values, Designer discards the values you specified after the run. If you want to save input values, create a launch configuration. For instructions, see "Creating a Launch Configuration for Running a Service" on page 404.

   Designer runs the service and displays the results in the Results view. If the launch configuration specifies an XML file to use as input, Designer submits the file to the server, which parses it into a node object and then passes it to the selected service.

Viewing Results from Running a Service

When you run a service, Designer displays the results in Results view.

The Results view contains a list of the recently run services and displays the results for the selected service.

You can specify the number of service results that Designer displays in the Results view using the **Number of results to display** option in the Results View preferences page.

Important:
If you are executing large documents multiple times, Software AG recommends that you set the **Number of results to display** option in the Results View preferences page to a smaller value, preferably 1, to ensure that the performance is not affected.

Note:
You can open the Results View preferences page by clicking the View Menu button (▽) and selecting **Preferences**.

For each service execution, the Results view can display the following tabs:

- **Messages** tab displays any messages from Designer about the launch configuration and any exceptions thrown by the service during execution.
- **Call Stack** tab identifies the flow step that generated the error and lists its antecedents; the **Call Stack** tab is only applicable to a flow service.
- **Pipeline** tab displays the contents of the pipeline at the time the service finished executing.

Messages Tab

The Messages tab in the Results view displays the time the launch configuration started and completed executing, the name and location of the launch configuration, and any error and exception messages that Integration Server generated during service execution. If you did not use a launch configuration when running or debugging the service, Designer displays the name and location of the launch configuration it created to execute the service.

Call Stack Tab

The Call Stack tab identifies which service generated the error and lists its antecedents. Call Stack also identifies the flow step that executed when the error occurred. Results view contains a Call Stack tab only for a flow service and only if the flow service throws an exception.

Keep the following points in mind when looking at the Call Stack tab.

- The call stack is LIFO based. That is, the top entry in the stack identifies the last (that is, most recent) service invoked. The bottom entry identifies the parent service (the one that you originally invoked from Designer). If the parent itself throws the exception, it will be the only entry in the call stack.

- You can go to the flow step that was executing when the error occurred by clicking ⇨ on the Results view toolbar. To view the service in Package Navigator view, right-click the service and select **Show In > Navigator.**

Pipeline Tab

The Pipeline tab in the Results view contains the contents of the pipeline after the service finishes executing.

Keep the following points in mind when examining the Pipeline tab in the Results view.

- The Pipeline tab shows *all* variables that the service placed in the pipeline, not just those that were declared in the service's input/output parameters.
- Variables that a service explicitly drops from the pipeline do not appear on the Pipeline tab.
- When you select a variable in the Pipeline tab, Designer displays details about the variable value in the details panel in the lower half of the Pipeline tab. For array variables, Designer displays the index number and position for each item in the array. You can copy and paste values from the details panel in the Pipeline tab.
- You can browse the contents of the Pipeline tab, but you cannot edit it directly. However, if you debug a flow service, you can edit the contents of the pipeline. For more information, see "Modifying the Flow Service Pipeline while Debugging" on page 444.
- You can save the contents of the Pipeline tab to a file and use that file to restore the pipeline at a later point. For additional information about saving and restoring the contents of the Pipeline tab in the Results view, see "Saving the Results" on page 423 and "Restoring the Results" on page 424.
- When a failure occurs within a Java service, the Pipeline tab represents the state of the pipeline at the point when that Java service was initially called. If the Java service made changes to these values before throwing the exception, those changes will note be reflected on the Pipeline tab.
- If you run a flow service and an error occurs, the Pipeline tab will only show results up to the point of the error.
- Variables with object types that Designer does not directly support will appear in the Pipeline tab, but because Designer cannot render the values of such objects, a value does not appear in the **Value** column. Instead, the **Value** column displays the object's Java class message.
- Variables that contain com.wm.util.Table objects appear as Document Lists in the Pipeline tab.

Saving the Results

You can save the results pipeline to a file. You might do this so that you can compare saved result pipelines to each other.

When you save a pipeline, it is saved in a file in XML format. The file you create can be used to:

- Dynamically load the pipeline at run time in a flow service by using the pub.flow:restorePipelineFromFile service.
- Load a set of input values when creating a launch configuration or when running or debugging a service.

You can view a pipeline file with an ordinary text editor. When saving the pipeline, keep the following points in mind:

- Only XML-codable variables are saved. This includes, Strings, String Lists, String Tables, Documents, and Document Lists. Variables that are not XML codable are not saved.
- Empty variables and null variables are saved.

Use the following procedure to save the results pipeline to a pipeline file.

To save the results pipeline

- On the Pipeline tab in the Results view, do one of the following:

Click...	To...
	Save the pipeline to your local file system. Specify a location and name for the file in the **Save As** dialog box. Click **Save**.
	Save the pipeline to *IntegrationServer_directory*\instances*instance_name*\pipeline directory on the machine on which Integration Server resides. In the **Save Pipeline to *serverName*** dialog box, specify the name for the file containing the pipeline contents.

Restoring the Results

When you load a pipeline file into the Pipeline tab in the Results view, the contents of the pipeline file completely *replaces* the results pipeline.

To restore service results

- On the Pipeline tab in the Results view, do one of the following:

Click...	To...
	Restore the pipeline from a file in your local file system. In the **Open** dialog box, navigate to and select the file. Click **Open**.

Click...	To...
	To restore the pipeline from the *IntegrationServer_directory*\instances*instance_name*\pipeline directory on the machine on which Integration Server resides. In the **Restore Pipeline from *serverName*** dialog box, specify the name for the file containing the pipeline contents.

Running Services from Results view

You can re-execute a launch configuration or a flow service using the same set of inputs as before from the Results view.

To rerun services from Results view

- In the Results view, select the service or element that you want to re-execute and click on the Results view toolbar. Integration Server re-executes the service or element using the same set of inputs as before.

Removing the Results from Results View

You can delete a result or all the results from the Results view.

To remove results from Results view

- In the Results view, do one of the following:

Click...	To...
	Remove the selected result from the Results view.
	To remove all results from the Results view.

Pinning a Result to Results View

You can pin a result to the Results view. This ensures that the particular result is available for your reference in the Results view even after several other service executions. You can unpin the result from Results view when you no longer need to refer to it. When you pin a result, the next time you execute a service, Designer shows the results in a new Results view.

When you pin a result to the Results view, Designer closes the pane that displays the list of the recently run services and displays only the pinned result. You will not be able to perform operations such as re-executing the service or element from Results view and removing service results from Results view.

To pin a result to Results view

- In the Results view, select the service result that you want to pin and click on the Results view toolbar. You can unpin a result from Results view by clicking the same button.

Sorting Results by Element Names in Results View

By default, Designer sorts the results in the Results view by timestamp. You can sort the results alphabetically by element names. Even if the results are sorted alphabetically, Designer uses timestamp information to remove the results from Results view to limit the number of results displayed in the Results view to the value specified in the **Number of results to display** preferences.

To sort results by element names

- On the Results view toolbar, click to sort the results by element names instead of timestamp. To sort the results by timestamp, click the same button again.

Running Services from a Browser

You can use the **Run in Browser** command to run a service from a browser (that is, to simulate a browser-based client). When you use this command, Designer prompts you for the service's input values, builds the URL necessary to invoke the service with the inputs you specify, and then passes the URL to your browser. When you use this command to run a service, your browser (not Designer) actually invokes the service and receives its results.

If you are developing services that will be invoked by browser-based clients, particularly ones whose output will be formatted using output templates, you will want to test those services using the **Run in Browser** command to verify that they work as expected.

To run a service using a browser as the client

1. In Package Navigator view, select the service you want to run.
2. In Designer: **Run > Run As > Run in Browser**

   If you have unsaved edits, Designer prompts you to save them.
3. If the service has input parameters, type the input values for each variable in the Input dialog box or click the **Load** button to retrieve the values from a file. For more information, see "Entering Input for a Service" on page 406.

   Note:Run in Browser only submits String and String List inputs to the service. If you want to pass other types of inputs, use the **Run > Run As > Run Service** option or set the values in the service instead of entering it in the **Enter Input for *serviceName*** dialog box.

4. If you want to pass empty variables (variables that have no value) to the service, select the **Include empty values for String Types** check box. When you select this option, empty Strings are passed with a zero-length value. If you do not select this option, Designer excludes empty variables from the query string that it passes to the browser.

5. If you want to save the input values that you have entered, click **Save**. Input values that you save can be recalled and reused in later tests. For more information about saving input values, see "Saving Input Values" on page 420.

6. Click **OK**. Designer builds the URL to invoke the service with the inputs you have specified, launches your browser, and passes it the URL.

   - If the service executes successfully, the service results appear in your browser. If an output template is assigned to the service, the template will be applied to the results before they are returned.
   - If the service execution fails with an error, an error message is displayed in the browser.

22 Debugging Flow Services

- About Debugging Flow Services 430
- Creating Launch Configurations for Debugging Flow Services 432
- Debugging a Flow Service 434
- Stepping Through Flow Services 435
- Using Breakpoints When Debugging Flow Services 439
- Disabling and Enabling Flow Steps and Transformers 442
- Disabling and Enabling Conditions 443
- Modifying the Flow Service Pipeline while Debugging 444
- Saving and Restoring the Flow Service Pipeline while Debugging 447
- Viewing Service Results from a Flow Service Debug Session 449
- Using the Server Log for Debugging 449

Designer provides a range of tools to assist you during the debugging phase of development. For example, you can:

- Run flow services, specify their input values, and inspect their results.
- Examine the call stack and the pipeline when an error occurs.
- Execute services in debug mode, a mode that lets you monitor a flow service's execution path, execute its steps one at a time, specify points where you want to halt execution, and examine and modify the pipeline before and after executing individual flow service steps.

About Debugging Flow Services

When you debug a flow service, Designer provides tools that you can use to monitor the execution path of the flow service and examine the service pipeline at various points during service execution. While debugging, you can also use Designer to save or modify the current pipeline.

About Debug Sessions

A debug session starts when you use a launch configuration to debug a service. Designer requires a launch configuration for debugging a flow service. If a service does not have a launch configuration, Designer creates one automatically. Designer saves the launch configuration to your workspace.

Designer creates a debug session for each launch configuration that you use for debugging a service. You can use multiple launch configurations simultaneously and have multiple debug sessions for the same launch configuration. This translates to multiple debug sessions for the same service. Debug view displays all of the debug sessions. The name of the launch configuration used for a debug session appears as a top level node in the Debug view. The launch configuration can appear in the Debug view multiple times, once for each debug session.

Once started, debug sessions can suspend, resume, or terminate.

A debug session suspends for the following reasons:

- The launch configuration specifies that the debug session should stop at the first flow step.
- A step command finishes execution (**Step Over**, **Step Into**, or **Step Return**). Designer suspends the debug session immediately before executing the next step in the flow service when a breakpoint is encountered. In case of **Debug to Here**, the execution continues till the specified step even if breakpoints are encountered.

A debug session *resumes* when you select **Run > Resume** or execute a step command.

A debug session *terminates* for the following reasons:

- The flow service that you are debugging executes to completion (error or success).
- You select the **Step Over** command for the last step in the flow service.
- You forcefully terminate the debug session by selecting **Run > Terminate**.

- You exit Eclipse.

About the Debug Perspective

When you debug a service, use the Debug perspective. The Debug perspective contains various views for helping you debug your service. The following table briefly describes the views available in the Debug perspective. With the exception of Results view, these views are part of the Eclipse Debugging framework.

View	Description
Debug view	Displays the debug sessions and contains tools to manage the debugging. When debugging a service in Designer, the Debug view displays the stack frames associated with each launch configuration. Debug view contains commands to start, stop, and step through a service.
Breakpoints view	Displays all the breakpoints currently set in your workspace.
Results view	Displays the results of running or debugging a service via a launch configuration in Designer.
Variables view	Displays information about the set of variables for the selected stack frame in Debug view. When using Designer to debug a service, Variables view displays the contents of the pipeline prior to the execution of the flow step in the selected stack frame in Debug view. The details pane in Variables view displays detailed information about the selected variable. You can edit the variable value in the detail pane. You can save or modify the contents of Variables view before resuming execution. Variables view will be blank after the service executes to completion.

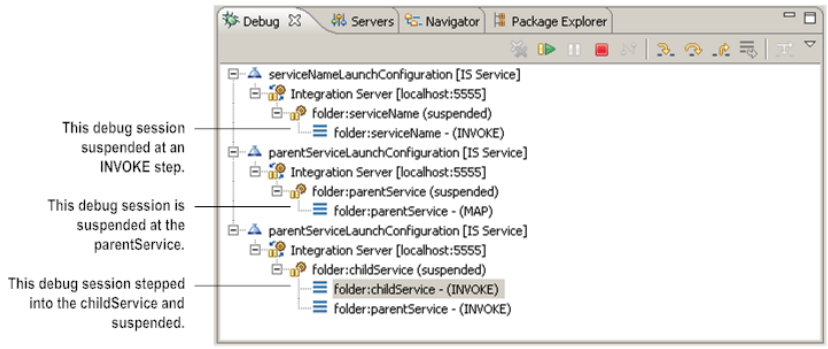
About Debug View

When debugging a service, Designer displays the debugging progress and the flow service editor (including the flow step that is about to be executed) in Debug view. Keep the following information in mind when looking at Debug view while debugging a flow service:

- For each launch configuration that you use to debug a service, Debug view contains a launch configuration stack frame.
- The launch configuration stack frame contains an Integration Server thread stack frame. The launch configuration can appear in the Debug view multiple times, once for each debug session.
- The Integration Server server thread stack frame contains the service thread stack frame.
- If a debug session is suspended, the service thread stack frame displays “(suspended)” after the service name.

- The service thread stack frame contains the name of the flow service and the step at which the debug session is suspended.
- If you stepped into a child INVOKE service or into a transformer in a MAP step, Debug view displays the parent service (and its flow step) and the child service (and its flow step) under the service thread stack frame. Designer displays the child service above the parent service because the child service is at the top of the call stack structure. Designer highlights the child step at which the debug session is suspended. Variables view displays the contents of the pipeline that will be passed to the child service.
- If you stepped into a MAP step to invoke a transformer, under the service thread stack frame, Designer displays MAPINVOKE before Designer executes the transformer. Designer displays MAPCOPY right after the service executes but before Designer executes the links from the transformer to the variables in **Pipeline Out.**

Debug view with three debug sessions



Creating Launch Configurations for Debugging Flow Services

You can use the same launch configuration to run and debug flow services. When using a launch configuration to debug a service, Designer stops at the first flow step (the default) or executes the service until it encounters a breakpoint.

Designer requires launch configurations to debug flow services. However, if a flow service does not have an associated launch configuration and you bypass the Debug Configurations dialog boxes when debugging the service, Designer creates one on the fly and saves it locally. You can use this configuration from one session to the next. In fact, Designer reuses this configuration every time you debug the service without creating a launch configuration.

By default, Designer saves launch configurations in an unexposed location of your workspace. However, you might want to share launch configurations with other developers. You can specify that Designer save a launch configuration to a shared file. On the **Common** tab in the Debug Configurations dialog box, select the **Shared file** option and provide a workspace location in which to save the file.

You might consider creating a launch configuration for each set of data that you routinely use to debug your service. This will provide you with a ready-made set of test cases against which to verify the service when it is modified by you or other developers in the future. Many sites establish a workspace project directory just for holding sets of test data that they generate in this manner.

Use the following procedure to create a launch configuration for use in debugging a flow service.

To create a launch configuration for debugging a flow service

1. In Designer: **Run > Debug Configurations**

2. In the **Debug Configuration** dialog box, select **IS Service** and click to add a new launch configuration.

3. In the **Name** field, specify a name for the launch configuration

4. On the **Service** tab, in the **Integration Server** list, select the Integration Server on which the service for which you are creating a launch configuration resides.

5. In the **Service** field, enter the name of the service for which you are creating a launch configuration of click **Browse** to select the service.

6. If you want Designer to stop at the first flow step when using the launch configuration, select the **Stop at first flow step** check box.

   If you clear the **Stop at first flow step** check box, Designer executes the service until the service ends or hits a breakpoint. If you want to debug the service by stepping through one flow step at a time, select the **Stop at first flow step** check box.

7. If you want Designer to pass the service an IData that contains input values for each input variable in the service signature, do the following:

   a. On the **Input** tab, select **Use IData**.

   b. Specify the input values to save with the launch configuration by doing one of the following:

      - Type the input value for each service input parameter. For more information about providing input values, see "Entering Input for a Service" on page 406.
      - To load the input values from a file, click **Load** to locate and select the file containing the input values. If Designer cannot parse the input data, it displays an error message indicating why it cannot load the data. For more information about loading input values from a file, see "Loading Input Values" on page 420
      - To load input values from a file and replace the service input signature with the structure and data types in a file, click **Load and Replace**.

   c. If you want to pass empty variables (variables that have no value) to the service, select the **Include empty values for String Type** check box. When you select this option, empty strings are passed with a zero-length value. If you do not select this option, Designer excludes empty value from the IData it passes to the service as input.

8. If you want Designer to send the flow service an XML document a input, do the following:

a. Select **Use XML**.

b. In the **Location** field, enter the path and file name of the XML document to use as input or click **Browse** to locate and select the XML file.

   Designer displays the contents of the XML document on the Input tab.

9. If you selected the **Use IData** option and you want to save the input values that you have entered, click **Save**. Input values that you save can be recalled and reused in later tests.

10. Click **Apply**.

11. If you want to execute the launch configuration, click **Debug**. Otherwise, click **Close**.

Debugging a Flow Service

While debugging a flow service, you can:

- Execute a flow service one flow step at a time and view the results of each step.
- Set breakpoints to specify points in a flow service at which processing should stop.
- Change the values passed to each step in the flow service.

Use the following procedure to debug a flow service.

To debug a flow service

1. In Package Navigator view, select the service you want to debug.

2. In Designer: **Run > Debug As > Debug Flow Service**

3. If multiple launch configurations exist for the service, use the **Select Launch Configuration** dialog box to select the launch configuration that you want Designer to use to debug the service.

4. If the launch configuration is set up to prompt the user for input values or there is no launch configuration, in the **Enter Input for *serviceName*** dialog box, specify input values for the service. Click OK. For more information about supplying input values, see "Entering Input for a Service" on page 406.

   Designer does one of the following:

   - If the launch configuration specifies that execution should stop at the first flow step when debugging, Designer prompts you to switch to the Debug perspective. Designer suspends flow service execution immediately before executing the first flow step. For more information about stepping through a flow service, see "Stepping Through Flow Services" on page 435.

- If you are not using an existing launch configuration to debug the service (that is, Designer created one for you automatically), Designer suspends flow service execution immediately before executing the first flow step.
- If the launch configuration does not stop a the first flow step, Designer executes the flow service until a breakpoint hit occurs. Designer prompts you to switch to the Debug perspective. To resume service execution after Designer encounters a breakpoint, select **Run > Resume**.
- If the flow service does not stop at the first flow step and does not contain a breakpoint, Designer executes the flow service to completion.

5. To execute flow steps one after another up to a specified flow step, right click and select **Debug to Here**.

Stepping Through Flow Services

You use the **Step Over**, **Step Into**, **Debug to Here**, and **Step Out** commands during a debug session to interactively execute a flow service one flow step at a time. Stepping through a flow is an effective debugging technique because it allows you to examine (and optionally modify) the data in the pipeline before and after each step. Additionally, if you are trying to isolate an error, step mode can quickly help you pinpoint the offending flow step.

If you want to...	Use...
Execute the current flow step	**Step Over**
Open a child flow a so that you can debug the individual flow steps within it	**Step Into**
Execute flow steps one after another up to a specified flow step	**Debug to Here**
Return to the parent flow from a child that you have stepped into	**Step Return**

Stepping Through a Flow Service

When stepping through a flow service, keep the following points in mind:

- To step through a top-level service, you must have Execute, Read, and List access to the service. To step through all the services within a top-level service, you must have Execute, List, and Read access to all services that the top-level service invokes.
- If the debug launch configuration is not configured to stop at the first flow step or there is not an enabled breakpoint in the flow service or one of its child services, Designer will execute the service to completion and will not suspend at any flow step.
- When you step through a flow step, Designer executes the step and then suspends the debug session.

- **Debug to Here** skips all the breakpoints and stops execution at the selected step based on the execution path, else execution completes.
- The Variables view displays the pipeline that will be passed to the next step in the flow service. You can modify, save, or restore the data while debugging the service.
- If you step over a flow step that contains an enabled breakpoint, Designer suspends service execution at the breakpoint.
- When you run a flow service in debug mode, Designer considers every **Run**, **Step Into**, and **Step Over** action to be a new service execution. When running a service in debug mode, the service execution ends when Designer encounters a breakpoint or the flow terminates. For a step command such as **Step Into** and **Step Over**, the service execution starts and end with each step command.
- Debugging a flow service that performs an asynchronous request/reply where Universal Messaging is the messaging provider has some unique considerations due to the nature of debugging in Designer and the request/reply channel that the publishing service uses to retrieve an asynchronous reply. For more information, see the *Publish-Subscribe Developer's Guide*.

To step through a flow service

1. Debug the service as described in "Debugging a Flow Service" on page 434.
2. In Debug view, select the flow step in the debug session for the service that you want to step through.
3. In Designer, select **Run > Step Over** or click on the Debug view toolbar.

   Designer executes the current step and then stops.
4. In Designer, select **Debug to Here** from the context menu.

   Designer executes the steps till the specified flow step and then stops.

Stepping Into and Out of a Child Service

Many times, the flow service you are debugging invokes other flow services (child services). In these cases it is useful to step through the individual flow steps within a child service, too. You do this with the **Step Into** and **Step Return** commands.

To step into and out of a child flow

1. Debug the service as described in "Debugging a Flow Service" on page 434.
2. In Debug view, step to the flow step that invokes the child flow service.
3. In Designer, select **Run > Step Into** or click on the Debug view toolbar.

Designer opens the child flow service and selects (but does not execute) the first step.

4. To execute the first step in the child flow service, select **Run > Step Over** or click on the Debug view toolbar. Repeat this step for each flow step that you want to individually execute within the child flow service.

5. If you want to return to the parent flow service without stepping through the entire child, select **Run > Step Return** or click on the Debug view toolbar. Designer executes the remaining steps in the child flow service, returns to the parent, and then selects (but does not execute) the next step in the parent flow.

Notes:

- If you select **Step Over** on the last step in the child flow service, Designer automatically returns you to the parent.
- You can use **Debug to Here** to execute up to particular step in the child flow service. Here, Designer skips all the breakpoints and stops execution at the selected step.
- You can use **Step Into** to step into a child flow that is nested within a child that you have stepped into.
- If you select **Step Return** without executing the entire child flow service and the child flow service subsequently contains an enabled breakpoint, Designer stops debugging when it hits the breakpoint.

Stepping Into and Out of a MAP Step

You can use the step commands to debug individual transformers within a MAP step.

To step into and out of a MAP step

1. Debug the service as described in "Debugging a Flow Service" on page 434.

2. In Debug view, step to the MAP flow step.

3. In Designer, select **Run > Step Into** or click on the Debug view toolbar.

   In Pipeline view, Designer highlights the links to the transformer (but does not execute) a transformer in the MAP step. Keep in mind that transformers in a MAP step are independent of each other and do not execute in a specific order.

4. If the transformer is a flow service and you want to step into the transformer, do the following:

   a. Select **Run > Step Into** or click on the Debug view toolbar. Designer executes the links to the transformer and steps into the transformer (flow service). Continue stepping through the transformer using Step Into or Step Over.

To return to the MAP step pipeline, select **Run > Step Return** or click on the Debug view toolbar.

b. Select **Run > Step Over** or click to execute the links between the transformer to the variables in **Pipeline Out**.

c. Repeat the above steps each transformer that you want to individually execute in the MAP step. If you want to execute the next transformer without stepping through transformer and link execution, select **Run > Step Over**.

5. If the transformer is not a flow service or you do not want to step into it, select **Run > Step Over** or click on the Debug view toolbar.

6. If you want to return to the parent without stepping through the entire MAP, select **Run > Step Return** or click on the Debug view toolbar. This executes the remaining transformers in the MAP, returns to the parent flow service, and selects (but does not execute) the next step in the parent flow service.

Note:

- If you select **Step Return**, Designer executes the remaining steps in the MAP and returns to the parent automatically. However, Designer stops executing if it encounters an enabled breakpoint.
- You can use **Step Into** to step into a transformer that is not a flow service.
- In Debug view, under the service thread stack frame, Designer displays MAPINVOKE before Designer executes the transformer. Designer displays MAPCOPY right after the service executes but before Designer executes the links from the transformer to the variables in **Pipeline Out**.

Stepping Into and Out of a ForEach Mapping

You can use the step commands to debug an individual mapping step within a ForEach mapping.

To step into and out of a ForEach Mapping

1. Debug the service as described in "Debugging a Flow Service" on page 434.

2. In Debug view, step to the ForEach mapping.

3. In the selected ForEach mapping, do the following:

a. Select **Run > Step Into** or click on the Debug view toolbar. Designer opens the ForEach mapping and selects, but does not execute, the first step.

b. To execute the first step within the ForEach mapping, select **Run > Step Over** or click on the Debug view toolbar. Repeat this step for each step that you want to individually execute

within the ForEach mapping. The step within can be a link, transformer, or a nested ForEach mapping.

c. To return to the parent ForEach mapping, select **Run > Step Return** or click on the Debug view toolbar.

Note:
In the ForEach mapping, if you do not want to step into it a particular step, select **Run > Step Over** or click on the Debug view toolbar.

4. If you want to return to the parent without stepping through the entire ForEach mapping, select **Run > Step Return** or click on the Debug view toolbar. This executes the remaining steps (links, transformers, or nested ForEach mappings) in the ForEach mapping, returns to the parent, and selects (but does not execute) the next step in the parent flow service or ForEach mapping.

Note:
If you select **Step Return**, Designer executes the remaining steps in the ForEach mapping and returns to the parent automatically. However, Designer stops executing if it encounters an enabled breakpoint.

Related Topics

"Stepping Into and Out of a MAP Step" on page 437

Using Breakpoints When Debugging Flow Services

Within Designer, a breakpoint is a point in a flow service where you want processing to halt when you debug that flow service. Breakpoints can help you isolate a section of code or examine data values at a particular point in the execution path. For example, you might want to set a pair of breakpoints before and after a particular segment of a flow so that you can examine the pipeline in the Variables view before and after that segment executes.

When you execute a service that contains a breakpoint or call a child service that contains a breakpoint, the service is executed up to, but not including, the designated breakpoint step. At this point, processing stops and the debug session suspends. To resume processing, you can execute one of the step commands or select **Run > Resume**. After you resume the debug session, Designer stops at any subsequent breakpoints.

When working with breakpoints, keep the following points in mind:

- Breakpoints are persistent in Designer.
- Breakpoints are also local to *your* Designer workspace. Breakpoints that you set in your workspace do not affect other developers or users who might be executing or debugging services in which you have set breakpoints.
- Breakpoints are only recognized when you execute a service in debug session. Breakpoints are ignored when you run a service.

- To set a breakpoint in a service, you must have Read access to a service. However, if the service is invoked within another service (a top-level service) to which you have Read access, you can set a breakpoint on the service within the top-level service.
- When you delete a flow step or transformer that contains a breakpoint, Designer removes the breakpoint.
- You can use breakpoints as markers in your flow services. To do this, assign a breakpoint to the flow step that you want to use as a marker. In Breakpoints view, you can quickly go to the flow step by right-clicking the breakpoint and selecting **Go to File** or by double-clicking the breakpoint.
- Breakpoints can be used in flow services that contain transactions, however, the breakpoint must be set before the transaction starts or after the transaction commits or rolls back. Do not set breakpoints within the transaction. If you do so, transactionality will not be honored and the flow service may throw an exception.
- When using breakpoints in a flow service that performs an asynchronous request/reply with webMethods messaging where Universal Messaging is the messaging provider, insert the breakpoint at or before the service that initiates the request and/or after the service that retrieves the request. That is, insert the breakpoint at or before the invocation of pub.publish:publishAndWait or pub.publish:deliverAndWait and/or after the invocation of pub.publish:waitForReply. Do not set a breakpoint on a step that occurs after the initiating the request but before retrieving the reply. When you run a flow service in debug mode, Designer considers every **Run**, **Step Into**, **Step Over** action to be a new service execution. The request/reply channel created by the publishing service is removed when Designer encounters the breakpoint. Setting a breakpoint after the publish but before retrieving the reply results in the removal of the request reply/channel which means that there is no channel from which the pub.publish:waitForReply service can retrieve a reply document.
- You can use Breakpoints view to manage your existing breakpoints.
- You can import/export breakpoints from one workspace to share them with other developers.

Breakpoint States

Breakpoints can have the following states.

Icon	State	Description
	Enabled/Non-Execution	The breakpoint is enabled, but there is no debug session in progress.
	Disabled/Non-Execution	The breakpoint is disabled and there is no debug session in progress.
	Skip/Enabled	Indicates the breakpoint is enabled but will be skipped.
	Skip/Disabled	Indicates the breakpoint is disabled and will be skipped.

Setting and Removing Breakpoints on Flow Step

You can set or remove a breakpoint on a flow step by toggling the breakpoint. During debugging, processing will halt immediately *before* this flow step.

To toggle a breakpoint on a flow step

- Open the flow service in which you want to set a breakpoint and do one of the following:
  - On the Tree tab, right-click the step at which you want to set the breakpoint and select **Toggle Breakpoint**. Designer displays or removes in the vertical margin next to the flow step.
  - On the Tree tab, right-click in the vertical margin next to the flow step at which you want to set a breakpoint and select **Toggle Breakpoint**. Alternatively, double-click in the margin next to the flow step. Designer displays or removes in the vertical margin next to the flow step.
  - On the Layout tab, double-click the connection line in front of the flow step at which you want to set a breakpoint and select **Toggle Breakpoint**. Designer displays or removes right before the flow step.
  - On the Layout tab, select the connection line in front of the flow step at which you want to set a breakpoint. Right-click and select **Toggle Breakpoint**. Designer displays or removes right before the flow step.

Note:
You can also use Breakpoints view to remove breakpoints or select **Run > Remove All Breakpoints**.

Setting and Removing Breakpoints on a Transformer

You can set a breakpoint on a transformer in a MAP step. When you execute a service that contains a breakpoint or calls a service that contains a breakpoint on a transformer, the service is executed up to, but not including, the designated breakpoint transformer.

Transformers in a MAP step execute in an arbitrary order. You cannot assume an order of execution. Consequently, some of the transformers in the MAP step might execute before Designer reaches the breakpoint, even if the transformers appear below the breakpoint in the Pipeline view. Likewise, transformers above the breakpoint might *not* execute before the breakpoint is encountered and the debug session suspends. These will execute when the debug session resumes.

You set or remove a breakpoint on a transformer by toggling the breakpoint state.

To toggle a breakpoint on a transformer

1. Open the flow service in which you want to set a breakpoint.

2. In the editor, select the MAP step containing the transformer that will function as the breakpoint.

3. In Pipeline view, right-click the name of the transformer that will function as the breakpoint and select **Toggle Breakpoint**. Designer displays ● or removes next to the transformer name.

Enabling and Disabling Breakpoints in a Flow Service

You can enable or disable a breakpoint to instruct Designer to stop at or ignore the breakpoint.

To enable or disable a breakpoint on a flow step or transformer

1. Open the flow service and navigate to the breakpoint. If a breakpoint is set on a transformer, select the MAP step and open Pipeline view.

2. Do one of the following:

   - To disable a breakpoint, right-click the breakpoint and select **Disable Breakpoint**.
   - To enable a breakpoint, right-click the breakpoint and select **Enable Breakpoint**.

Note:
You can also enable and disable breakpoints using Breakpoints view.

Skipping Breakpoints in a Flow Service

You can use Designer to change the breakpoint state of all breakpoints to "skip". Designer does not execute breakpoints with a skip state regardless of whether the breakpoint is enabled or disabled. Designer debugs services as if the breakpoints did not exist or were disabled. By instructing Designer to skip all breakpoints, you can debug the service without halting execution for a breakpoint without removing or changing the enabled/disabled state of the breakpoint.

To skip all breakpoints

- In Designer: **Run > Skip All Breakpoints**

Disabling and Enabling Flow Steps and Transformers

As part of debugging services, you can disable flow steps and transformers. Disabled flow steps and transformers do not execute. Disabling a step or transformer is useful in many debugging situations. For example, you might want to disable one or more steps to isolate a particular segment of a flow, similar to the way you might "comment out" a section of source code in a program you are debugging.

Keep the following points in mind when disabling and enabling flow steps and transformers:

- Disabling a step or transformer sets a persistent attribute that is saved in the flow service. Once you disable a step or transformer, it remains disabled until you explicitly re-enable it with Designer.
- Steps and transformers that you disable are not executed at run time.
- The ⊘ symbol appears next to disabled steps and transformers.
- If you disable a parent step (for example, a LOOP or a BRANCH), Designer disables its children automatically.
- If you disable a MAP step, Designer disables the transformers in the MAP step automatically.
- Disabling a step or transformer removes a breakpoint hosted by that step or transformer.

Important:
The run-time effect of disabling a step is the same as deleting it. Disabling a key step or forgetting to re-enable a disabled step or transformer can break the logic of a service and/or cause the service to fail. Designer allows you to disable any step or transformer in a flow service, but it is your responsibility to use this feature carefully.

To disable or enable a flow step or transformer

1. Open the flow service that you want to edit.
2. Do one of the following:
   - In the editor select the step that you want disable or enable.
   - In the editor select the MAP step containing transformers that you want to disable or enable.
3. Right-click the step or transformer and do one of the following:
   - Select **Disable Step** to disable the step or transformer.
   - Select **Enable Step** to re-enable the step or transformer.

Disabling and Enabling Conditions

When you link variables to each other, you can apply a condition to the link that connects the variables. At run time, this condition needs to be true for the value of the source variable to be copied to the target variable. During debugging, you might want to disable or remove the condition from the link to make sure that Designer properly copies data between variables. By disabling the condition, you instruct Designer to ignore the condition placed on the link and automatically execute the link.

Disabling the condition preserves the written expression. When you enable the condition, you do not need to rewrite the expression.

The Pipeline view uses a blue link (line) to indicate that properties (such as conditions and array indexes) have been applied to the link between variables. Designer retains the blue color even when you disable the applied condition to remind you that properties have been set.

To disable or enable a condition placed on a link between variables

1. Open the flow service that you want to edit.
2. In the editor, select the INVOKE or MAP step that contains the link with the condition you want to disable or enable.
3. In the Pipeline view, select the link with the condition that you want to disable.
4. In the **General** category of the Properties view, do one of the following:
   - To disable the condition, set the **Evaluate copy condition** property to **False**.
   - To re-enable the condition, set the **Evaluate copy condition** property to **True**.

Modifying the Flow Service Pipeline while Debugging

During debugging, you can modify the contents of the pipeline and submit those changed values to the next step in the flow service. For example, if you want to see the effect that different values for a variable have on the rest of the service, you can modify the values in the pipeline and continue debugging. You can also drop values from the pipeline. This functionality is useful for debugging.

When modifying the pipeline, keep the following points in mind:

- You can only modify the pipeline when a subsequent step in the service exists to which to pass the pipeline values. You cannot modify the values of the pipeline after the service ends. However, if you debug the service using the step commands, you can modify the pipeline values for the next flow step in the service.
- You cannot modify the values of unconstrained Objects and Object lists. However, you can drop them from the pipeline.
- You cannot modify the values of recursive documents at the top level. However, you can expand the document and set values at the individual element level.
- When you modify values or drop variables from the pipeline, the changes only apply to the current debugging session. The service is not permanently changed.
- You can only modify or drop existing variables. You cannot add new variables to the pipeline.
- You can load an entirely different pipeline for Designer to pass to the next step in the flow service.
- You can save the contents of the pipeline to a file. You may want to save the results of specific flow steps to a file to compare with other services or to use in later debug sessions.

Changing Variable Values

Keep the following points in mind when changing the values of variables while debugging the service:

- When you modify values in the pipeline, the changes only apply to the current debugging session. Neither the service nor launch configuration is permanently changed.
- You can only modify existing variables. You cannot add new variables to the pipeline.
- You cannot modify the values of unconstrained Objects and Object lists. However, you can drop them from the pipeline.
- You cannot modify the values of recursive documents at the top level. However, you can expand the document and set values at the individual element level.
- Variables that contain com.wm.util.Table objects appear as document lists in Variables view.
- You can edit rows and columns for String Table and Document List variables.
- You can use the step tools to step to the location in the service at which you want to change the pipeline. You can also set a breakpoint on the flow step at which you want to change the pipeline values.
- You can only change the pipeline for the top-most stack frame in the debug session.

To change the value of variable while debugging

1. Debug the service as described in "Debugging a Flow Service" on page 434.
2. In the debug session, use the step command or a breakpoint to reach the step for which you want to change variable values in the flow service.
3. In Variables view, right-click the variable whose value you want to change and select **Change Value**.

   The following table lists where you can find more information about how to update values based on the data type of the variable.

For this type of variable...	See...
String	"Specifying a Value for a String Variable" on page 408
String List	"Specifying Values for a String List Variable" on page 409
String Table	"Specifying Values for a String Table Variable" on page 411
Document Document Reference	"Specifying Values for a Document Variable that Has Defined Content" on page 413

For this type of variable...	See...
	"Specifying Values for a Document Variable with No Defined Content" on page 414
Document List Document Reference List	"Specifying Values for a Document List Variable" on page 416
Object	"Specifying a Value for an Object Variable" on page 418
Object List	"Specifying Values for an Object List Variable" on page 419

4. Continue debugging the service using the step commands or selecting **Run > Resume**.

Note:
You can also change a variable value by selecting the variable and then modifying the value in the detail pane.

Dropping Variables

When dropping variables from the pipeline while debugging the service, keep the following points in mind:

- You can only modify the pipeline when a subsequent step in the service exists to which to pass the pipeline values. You cannot modify the values of the pipeline after the service ends. However, if you debug the service using the step commands, you can modify the pipeline values for the next flow step in the service.
- When drop variables from the pipeline, the changes only apply to the current debugging session. The service is not permanently changed.
- You can only drop existing variables. You cannot add new variables to the pipeline.
- You can only change the pipeline for the top-most stack frame in the debug session.

To drop values from the pipeline while debugging

1. Debug the service as described in "Debugging a Flow Service" on page 434.

2. In the debug session, use the step command or a breakpoint to reach the step for which you want to drop a pipeline variable.

3. In Variables view, select the variable you want to drop from the pipeline and click on the Variables view toolbar. Designer removes the variable from Variables view.

4. Continue debugging the service using the step commands or selecting **Run > Resume**.

Saving and Restoring the Flow Service Pipeline while Debugging

Because the pipeline contains the data that a service operates against, the ability to save and restore the pipeline when you are debugging a service is something you may frequently want to do. For example, if a service is failing intermittently at run time, you may want to save the pipeline so you can capture and examine the data that it was running against after a failure.

Saving the Flow Service Pipeline while Debugging

You can save the pipeline to a file, which you can use to restore the pipeline to its current state at a later point in time. This is useful when you want to debug another service against the current set of pipeline values or if you want to restore the pipeline to this exact state later in the debugging process. There are three ways to save the contents of the pipeline:

- Manually save the contents when you debug a service using Designer.
- Automatically save the pipeline at run time using the **Pipeline debug** property. For more information about this property, see "Automatically Saving or Restoring the Pipeline at Run Time" on page 174.
- Programmatically save the pipeline at run time by invoking pub.flow:savePipelineToFile at the point where you want to capture the pipeline. For more information about using this service, see the *webMethods Integration Server Built-In Services Reference*.

When you save a pipeline, it is saved in a file in XML format. The file you create can be used to:

- Manually load the pipeline into Variables view while debugging a service.
- Automatically load the pipeline at run time using the **Pipeline debug** property.
- Load a default set of input values when creating a launch configuration.
- Load a set of input values into the Input dialog box when debugging a service with Designer.
- Dynamically load the pipeline at run time using the pub.flow:restorePipelineFromFile service.

Note:
When using MTOM streaming for SOAP attachments, *messageContext* variables and/or XOPObject fields will not be available in the saved pipeline. A *messageContext* variable is used by many pub.soap services to hold the SOAP message on which the service acts. XOPObject fields are Objects that use the com.wm.util.XOPObject Java wrapper type. For more information about MTOM Streaming, see the *Web Services Developer's Guide*.

Saving the Pipeline to a File while Debugging

When saving the pipeline during a debugging session, keep the following points in mind:

- Only XML-codable variables are saved. This includes, Strings, String lists, String tables, documents, and document lists. Variables that are not XML codable are not saved.

- Empty variables and null variables are saved.

To save the pipeline to a file while debugging

1. Debug the service as described in "Debugging a Flow Service" on page 434.

2. In the debug session, use the step command or a breakpoint to reach the step for which you want to save the pipeline.

3. Do one of the following:

   - To save the pipeline to your local file system, click on the Variables view toolbar. Specify a location and name for the file in the **Save As** dialog box. Click **Save**.
   - To save the pipeline to the *IntegrationServer_directory*\instances*instance_name*\pipeline directory on the machine on which Integration Server reside, click on the Variables view toolbar. In the **Save Pipeline to *serverName*** dialog box, specify the name for the file containing the pipeline contents.

4. Continue debugging the service using the step commands or selecting **Run > Resume**.

Restoring the Flow Service Pipeline while Debugging

Restoring a pipeline is useful when you simply want to inspect the values in a particular pipeline file (perhaps one that contains the pipeline from a failed service). Additionally, it is useful in many debugging situations. For example, you can use it to replace the existing pipeline with a different set of values when stepping though a flow service with the debugging tools.

There are three ways to restore the contents of the pipeline:

- Manually load the saved pipeline into the Variables view while debugging in Designer.
- Automatically load the saved pipeline at run time by using the **Pipeline debug** property.
- Programmatically load a saved pipeline at run time by invoking pub.flow:restorePipelineFromFile at the point where you want to modify the pipeline. For more information about using this service, see the *webMethods Integration Server Built-In Services Reference*.

Note:
When using MTOM streaming for SOAP attachments, *messageContext* variables and/or XOPObject fields will not be available in the saved pipeline. A *messageContext* variable is used by many pub.soap services to hold the SOAP message on which the service acts. XOPObject fields are Objects that use the com.wm.util.XOPObject Java wrapper type. For more information about MTOM Streaming, see the *Web Services Developer's Guide*.

Loading a Saved Pipeline while Debugging

When you load a pipeline file into Variables view, the contents of the pipeline file completely *replaces* the current pipeline. Designer passes the new set of values to the next step. If you want to

merge the contents of the file with the existing pipeline, use the pub.flow:restorePipelineFromFile service instead and set its *merge* parameter to "true."

To load a pipeline file into Variables view while debugging

1. Debug the service as described in "Debugging a Flow Service" on page 434.

2. In the debug session, use the step command or a breakpoint to reach the step for which you want to load the saved pipeline.

3. Do one of the following:

   - To load the pipeline from your local file system, click on the Variables view toolbar. In the **Open** dialog box, navigate to and select the file. Click **Open**.
   - To load the pipeline from the *IntegrationServer_directory*\instances*instance_name*\pipeline directory on the machine on which Integration Server reside, click on the Variables view toolbar. In the **Load IData from Server** dialog box, specify the name for the file containing the pipeline contents.

4. Continue debugging the service using the step commands or selecting **Run > Resume**.

Viewing Service Results from a Flow Service Debug Session

When you execute a service by debugging it, Designer displays the results in Results view. The Results view when debugging a flow service is the same as when running a service. For more information, see "Viewing Results from Running a Service" on page 422.

Using the Server Log for Debugging

Integration Server maintains a log file that contains information about activity on the server. By default, Integration Server creates one server log per day. The server log file resides in the following location:

Integration Server_directory \instances*instance_name*\logs

You can optionally redirect server log messages to the console rather than to a file by using the `-log` startup switch with a value of none. For more information about this switch, see information about starting Integration Server in *webMethods Integration Server Administrator's Guide*.

Content of the Server Log

The server log contains operational and error information that the server's major subsystems write to the log. For example, the package subsystem logs information into server log when it loads and unloads packages; the flow manager records information in the log when it processes a flow service; the HTTP port records requests that it receives, and so forth.

Additionally, you can code your service to log information that is useful for debugging. For information, see "Writing Information to the Server Log" on page 450.

Note:
The server logs exceptions thrown by individual services, to the error log. For more information about using the error log, see *webMethods Integration Server Administrator's Guide*.

Debug Level Defines What and How Much the Server Logs

To define the type and amount of information that the server logs, set the server's *debug level*. The debug level settings range from Off, indicating you want the server to log nothing, to Trace, indicating that you want the server to maintain an extremely detailed log.

Use the Integration Server Administrator **Settings > Logging > View Server Logger Details** screen to set debug levels that Integration Server uses for each of its facilities. When debugging an issue, you can use this screen to increase the logging level for a specific Integration Server facility. For example, you might set the logging level for the Services facility to Trace.

When you have not defined a specific debug level for a facility, Integration Server uses a default debug level. You configure the default by setting the logging level for the Default facility on the **Settings > Logging > View Server Logger Details** screen. Integration Server also uses the Default facility setting as the value of the watt.debug.level server configuration parameter. If you do not define a default debug level, Integration Server uses Info, which means the server logs informational, warning, error, and fatal messages.

When you start the server, you can temporarily override the default debug level by specifying an alternative level on the startup command. This setting remains in effect until you shutdown and restart the server.

For more information about the available debug levels, setting the debug level, and configuring server logging, see *webMethods Integration Server Administrator's Guide*.

Important:
Because debug levels above Info can produce lots of detail and can quickly generate an extremely large log file, do not run your server at the Debug or Trace levels except for brief periods when you are attempting to troubleshoot a particular issue.

Writing Information to the Server Log

Integration Server provides built-in services that allow you to write information to the server log at run time. These can be useful during debugging because you can use them to build signals that indicate whether certain segments of code were executed. You can also use them to record the run-time value of a specific variable.

There are two ways to write information to the server log at run time. You can:

- Write an arbitrary message to the log using pub.flow:debugLog.
- Dump the contents of the entire pipeline to the log using pub.flow:tracePipeline.

Writing an Arbitrary Message to the Log

To write an arbitrary message to the server log, invoke the pub.flow:debugLog built-in service. You can invoke pub.flow:debugLog from a flow service or a coded service (such as a Java service). When this service executes, it inserts a text string that you specify into the server log. You might use it to post progress messages at certain points in a service (to indicate whether certain segments of code were executed) or to record the value of a particular variable in the log file so you can examine it after the service executes. In the following example, the last two messages are progress messages that were posted to the server log using pub.flow:debugLog.

```
2012-03-28 16:56:12 EDT [ISS.0028.0005C] Loading LogDemo package
2012-03-28 16:56:53 EDT [ISC.0081.0001E] New LogDemo:demoService
```

```
2012-03-28 16:57:56 EDT [ISP.0090.0004C]
begin database update
```

```
2012-03-28 16:57:56 EDT [ISP.0090.0004C]
database update completed
```

To use pub.flow:debugLog to write an arbitrary message to the server log

1. In your service, invoke pub.flow:debugLog at the point where you want the service to write a message to the server log.

2. Set the following parameters:

Key	Description
message	A String that defines the message that you want written to server log. This can be a literal string. However, for debugging purposes, it is often useful to link this parameter to a pipeline variable whose run-time value you want to capture.
function	(Optional) A String that identifies your service as the component that posted the message to the log. When displaying the server log, the server lists the String you specify in the second column of the message. Assigning a value to *function* makes it easier for you to locate your service's message when you examine the server log. Although you can assign a text string of any length to *function*, the server displays only the first six characters. If you do not assign a value to *function*, debugLog omits the label.
level	(Optional) A String specifying the debug levels under which this message is to be recorded in the log. If the server is running at a debug level lower than the value set in *level*, the message is not recorded in the log file.

Key	Description
	If you do not specify *level*, the Fatal level is assumed, which means that the message is recorded in the log file regardless of which debug level the server is running at. For more information about debug level, see *webMethods Integration Server Administrator's Guide*.

3. Save the service. (If you are using your own IDE, you will need to recompile the service, register it again on Integration Server, and reload its package.)

4. Execute the service.

For additional information about pub.flow:debugLog, see the *webMethods Integration Server Built-In Services Reference*.

Dumping the Pipeline to the Log

Sometimes when you are debugging a service, it is useful to obtain a snapshot of the entire pipeline at a certain point in the service. You can do this by invoking pub.flow:tracePipeline, which puts a copy of the current pipeline in server log. You can invoke pub.flow:tracePipeline from a flow service or a coded service (such as a Java service).

The following example shows the start and end pipeline that was written to the server log with pub.flow:tracePipeline.

```
2012-03-28 17:37:10 EDT [ISP.0090.0001C] --- START tracePipeline
[3/28/12 5:37 PM] ---
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 0 filename = D:\Program Files\
Software AG\IntegrationServer\packages\Examples\pub\goes\catalogue.xml
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 0 Buyer = Caroline Wielman
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 0 Address =>
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 1 Street1 = 15788 Cedar Avenue
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 1 City = Apple Valley
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 1 State = MN
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 1 postalCode = 55124
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 0 Order =>
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 1 Date = 5/25/2002
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 1 Items[0] =>
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Code = 965003
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Description = MaxGear D LtWt D
Carabiner
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Qty = 300
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Price = 8.50
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Total = 2800
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 1 Items[1] =>
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Code = 896301
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Description = Hikes 10.5x50
Standard Rope
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Qty = 50
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Price = 175
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Total = 8750
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 1 Items[2] =>
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Code = 965007
2012-03-28 17:37:10 EDT [ISP.0090.0008C] 2 Description = MaxGear D Quick
```

```
Lock Carabiner
2002-05-28 17:37:10 EDT [ISP.0090.0008C] 2 Qty = 500
```

To use pub.flow:tracePipeline, take the following general steps

1. In your service, invoke pub.flow:tracePipeline at the point where you want the service to dump a copy of the pipeline to the server log.

2. Set the following parameters:

Key	Description
level	(Optional) A String specifying the debug levels under which the pipeline is to be written to the log. If the server is running at a debug level lower than the value set in *level*, the pipeline is not written to the log file. If you do not specify *level*, Fatal is assumed, which means that the pipeline is written to the log file regardless of which debug level the server is running at. For more information about debug level, see *webMethods Integration Server Administrator's Guide*.

3. Save the service. (If you are using your own IDE, you will need to recompile the service, register it again on Integration Server, and reload its package.)

4. Execute the service. For additional information about pub.flow:tracePipeline, see the *webMethods Integration Server Built-In Services Reference*.

Debugging Map Services

Debugging in map services is similar to that in flow service. Designer uses the same tools, commands, and processes to debug a map service as used in a flow service. For details on debugging a flow service, see "Debugging Flow Services" on page 429.

23 Debugging Java Services

- About Debugging a Java Service while its Class Runs in Designer 456
- About Test Harnesses 457
- About Java Application Launch Configuration 459
- How to Suspend Execution of a Java Class while Debugging 462
- Debugging a Java Service while its Class Runs in Designer 462
- About Debugging a Java Service while it Runs in Integration Server 465

Designer provides the ability to debug a Java service by debugging the Java class associated with the Java service maintained in the Service Development Project.

About Debugging a Java Service while its Class Runs in Designer

In Designer, the primary way to debug a Java service is to debug the Java class associated with the Java service that Designer maintains in a Service Development Project.

Note:
As a secondary method of debugging a Java service, you can debug a Java service that is running in Integration Server. This method requires setup on the Integration Server to change the way the server starts and that can affect the server's performance. For more information, see "About Debugging a Java Service while it Runs in Integration Server " on page 465.

The functionality that Designer provides to debug a Java service by debugging its Java class is an extension of the Eclipse Java Development Tools (JDT) debugger. The JDT debugger acts on Java classes that are in the local workspace; it cannot debug the Java service in Integration Server. As a result, to debug a Java service, you use the JDT debugger to debug the service's Java class that Designer maintains in a Service Development Project. Debugging the Java class might produce different results than when the Java service executes in Integration Server, depending on differences in JVM system properties, date/time, time zone information, locale, language settings, encodings, etc.

When debugging a Java service in this way, you can debug the primary method and shared code of the Java class that represents the Java service. To debug the Java class, you launch it in debug mode and use the JDT debugger to suspend/resume the execution of the Java class, inspect variables, and evaluate expressions.

The actions you take to use the debugger are:

- **Optionally set breakpoints** to identify locations where you want the debugger to suspend execution when running the Java class in debug mode. For more information, see "How to Suspend Execution of a Java Class while Debugging" on page 462.
- **Generate a test harness,** which is a Java class that you generate for the Java service you want to debug. The logic that Designer generates for the test harness sets up the inputs, invokes the Java class, and displays the outputs.
- **Optionally create a Java Application launch configuration** to configure settings for debugging the Java class. For example, you might want to set JVM arguments to match the settings Integration Server uses so that your test more closely matches how the Java service would execute in Integration Server. For more information, see "About Java Application Launch Configuration" on page 459. If you do not create a launch configuration, Designer creates one on the fly and saves it locally in an unexposed location of your workspace.
- **Launch the test harness in debug mode.** The test harness prompts for input values and then launches the Java class you want to debug in debug mode.

  By default, the debugger executes the Java class using the JRE in the Service Development Project where the Java service resides. You can change the Service Development Project's JRE

by updating the project's Java Build Path property. You can also specifically identify the JRE to use for debugging by identifying the JRE in the Java Application launch configuration.

If the Java class being debugged invokes a service, the invoked service runs in Integration Server. The debugger treats the statement to invoke a service like any executable line of code in the Java class; that is, you can Step Over it and see results from it. You cannot use the debugger to Step Into the invoked service.

If the debugger suspends execution of the service, Designer switches to the Debug perspective. The Debug view will show the test harness class and be positioned at the statement where the execution was suspended. You can use the other views in the Debug perspective to inspect the state of the Java service to this point. You can use the actions in the Debug view toolbar to resume the execution. For more information about suspending execution, see "How to Suspend Execution of a Java Class while Debugging" on page 462.

When the execution of the Java service completes, the debugger displays a window that contains the service results.

About Test Harnesses

A test harness is a Java class that you generate for a Java service you want to debug.

To debug a Java service in Designer, you use the Java Development Tools (JDT) debugger, which acts on Java classes in the local workspace. As a result, when debugging a Java service you actually debug the copy of a service's Java class that Designer maintains in a Service Development Project. However, to do so, you first need to generate a test harness for the Java class. The test harness sets the input parameters for the Java class you are debugging and then launches the Java class in debug mode.

When you generate a test harness, Designer stores the Java class for the test harness in the same Service Development Project and Java package where the Java class for the service being debugged resides. Designer uses the following format to name the test harness Java class:

<*serviceName*>_TestHarness.java

For example, if you generate a test harness for the service named "checkStatus", Designer assigns the test harness Java class the name "checkStatus_TestHarness.java".

When you launch the test harness, by default, it firsts prompts you to supply login credentials for Integration Server. The test harness must have login credentials so that it can connect to Integration Server to obtain the service's input signature. If the Java service has an input signature, the test harness then prompts you to supply input values. You can type in values or load values from a file. After the test harness has the input values, it executes the Java class you want to debug in debug mode. You can use the debugger to debug your Java class. When execution of the Java class completes, the test harness displays the outputs from the Java class in a popup window.

You can update the logic that Designer generates for a test harness to make the following modifications:

- Change the Integration Server to which the test harness connects.

By default, the test harness attempts to connect to the Integration Server used to create the test harness. You can specify a different Integration Server.

- Update the test harness to connect to Integration Server using SSL.

  By default, the test harness does not use SSL when connecting to Integration Server. You can uncomment logic in the generated test harness so that it uses SSL.

- Provide a user name and password for the Integration Server.

  Provide Integration Server credentials to prevent the test harness from prompting for the user name and password. This is useful if you plan to launch the test harness several times to debug a Java class. However, if you want to share the test harness with other users, do not supply your user name and password because this presents a security risk.

For instructions for how to generate a test harness, see "Creating a Test Harness" on page 458.

Creating a Test Harness

Use the following procedure to create a test harness for a Java class that you want to debug.

To create a test harness

1. In the Package Navigator view, right click the Java service for which you want to generate a test harness and select **Generate Code**.

   Alternatively, if the Java service is open in the editor, you can right click in the editor and select **Generate Code**.

2. In the Code Generation window, select **For debugging this service** and click **Finish**.

   Designer generates the test harness with a single main primary method and displays it in the JDT debugger editor.

3. Optionally update the Integration Server to which the test harness will connect to obtain the input parameters of the Java class to debug.

   a. Locate the following statements in the test harness Java class:

   ```
   // Connect to server - edit for alternate server
   String  server = "serverHost:portNumber";  //$NON-NLS-1$
   ```

   By default, the code identifies the Integration Server associated with the Java service for which you generated the test harness.

   b. Replace the host name and port number with the host name and port number of an alternate Integration Server.

4. If you want the test harness to use SSL when connecting to Integration Server:

a. Locate the following statements in the test harness Java class:

```
// To use SSL:
//
// context.setSecure(true);

// Optionally send authentication certificates
//
// String  cert    = "c:\\myCerts\\cert.der";  //$NON-NLS-1$
// String  privKey = "c:\\myCerts\\privkey.der";  //$NON-NLS-1$
// String  cacert  = "c:\\myCerts\\cacert.der";  //$NON-NLS-1$
// context.setSSLCertificates(cert, privKey, cacert);
```

b. Remove the `//` from the `context.setSecure(true)` statement to uncomment it.

c. If you want to send authentication certificates, update the certificates information and remove the `//` from the lines to uncomment them.

5. If you want to provide Integration Server credentials:

a. Locate the following statements in the test harness Java class:

```
// Set username and password for protected services
String username = null;
String password = null;
```

b. Specify a user name and password on these lines.

Important:
If you want to share the test harness with other users, do not supply your user name and password because this presents a security risk.

6. Select **File > Save** to save any changes you made.

About Java Application Launch Configuration

When debugging a Java class, the Java Development Tools (JDT) debugger requires a Java Application launch configuration. If you do not create one, Designer will automatically create one on the fly and save it locally. You can use this configuration from one session to the next. In fact, Designer reuses this configuration every time you debug the service without creating a launch configuration.

You cannot use the same launch configuration that you use to run a Java service. To run a Java service, you create an IS Service launch configuration, which defines settings for how the service is to run on Integration Server. For debugging Java services, you need a Java Application launch configuration, which defines settings for how the Java class executes in the Service Development Project.

The following lists the tabs available when creating a Java Application launch configuration and the type of information you specify on each:

- **Main tab.** Specify the name of the Service Development Project that contains the Java class you want to debug and the fully-qualified name of the Java class.

  Select the **Stop in main** check box if you want the debugger to suspend execution in the main method when you launch the Java class in debug mode. For more information, see "How to Suspend Execution of a Java Class while Debugging" on page 462.

- **Arguments tab.** Specify Program and JVM arguments to use when debugging. You might want to set JVM arguments to match the settings Integration Server uses so that your test more closely matches how the Java service would execute in Integration Server.

- **JRE tab.** Specifies the JRE to use when executing the Java class in debug mode. By default, it is set to the JRE in the Service Development Project. You can specify an alternative JRE to use when debugging.

- **Classpath tab.** Specifies the location of class files to use when executing the Java class in debug mode.

- **Source tab.** Specifies the location of source files to display in the Debug view. If you want to debug the source associated with any third-party jar files, you can specify them on this tab.

- **Environment tab.** Specifies the environment variable values to use when executing the Java class in debug mode.

- **Common tab.** By default, Designer saves launch configurations to an unexposed location of the workspace. However, you might want to share launch configurations with other developers. You can specify that Designer save a launch configuration to a shared file using the **Shared file** option and providing a workspace location in which to save the file.

Creating a Java Application Launch Configuration

Use the following procedure to create a Java Application launch configuration.

To create a Java Application launch configuration

1. In Designer: **Run > Debug Configurations**

2. In the Debug Configurations dialog box, select **Java Application** and click to add a new launch configuration.

3. On the **Main** tab in the **Project** field, specify the Service Development Project that contains the test harness and the Java class that you want to debug.

4. In the **Main Class** field, specify the fully-qualified name for the test harness Java class.

5. Select the **Stop in main** check box if you want the debugger to suspend execution in the main method of the test harness when you launch it in debug mode. For more information, see "How to Suspend Execution of a Java Class while Debugging" on page 462.

6. On the **JVM** tab, optionally specify any Program or JVM arguments you want to use.

7. On the **JRE** tab, specify whether to use the default JRE or an alternate JRE that you identify.

   - To use the default JRE, that is the JRE in the Service Development Project, select **Project JRE**.
   - To identify an alternate JRE, select **Alternate JRE** and click **Installed JREs** to select the JRE to use.

8. On the **Classpath** tab, optionally add, re-arrange, or remove the class files to use when executing the Java class in debug mode.

9. On the **Source** tab, optionally update the source lookup path.

10. On the **Environment** tab, optionally add environment variables.

11. On the **Common** tab, if you want to share this launch configurations with other developers, select the **Shared file** check box and specify a workspace location in which to save the configuration.

12. Click **Apply**.

13. Click **Debug** to immediately launch the test harness Java class in debug mode using this launch configuration, or click **Close**.

Updating a Java Application Launch Configuration

You can update any existing Java Application launch configuration, including those that Designer created on the fly. Use the following procedure to update an existing Java Application launch configuration.

To update a Java Application launch configuration

1. In Designer: **Run > Debug Configurations**

2. In the Debug Configurations dialog box, under **Java Application** select the launch configuration you want to edit.

   The panels on the right display the information for the selected launch configuration.

3. Update the information for the launch configuration. For more information on what to specify, see "Creating a Java Application Launch Configuration" on page 460.

4. Click **Apply**.

5. Click Debug to immediately launch the test harness Java class in debug mode using this launch configuration, or click **Close**.

How to Suspend Execution of a Java Class while Debugging

When you launch a Java class in debug mode, it runs until the execution is suspended. When the execution is suspended, Designer switches to the Debug perspective, allowing you to use the views in the perspective to inspect the state of the Java class. If the execution is never suspended, the Java class runs from beginning to end without switching to the Debug perspective, and as a result, never allowing you to inspect the state of the Java class in the middle of execution.

To suspend the Java class, you must do at least one of the following *before* launching the Java class in debug mode:

- **Set breakpoints in the Java class.**

  To set breakpoints, open the Java service in the Java service editor and double click in the left margin next to the executable lines of code where you want breakpoints.

  Designer displays the breakpoint enabled icon () in the margin.

  When you launch the Java class in debug mode and the debugger encounters a breakpoint, it suspends execution. At that time, if the Debug perspective is not already in use, Designer switches to it. The Debug view will show the test harness class and be positioned at the statement for which you created the breakpoint.

- **Select "Stop in main" in the Java Application launch configuration.**

  Create or update the Java Application launch configuration that you want to use for debugging the Java class so that the **Stop in main** check box is selected on the **Main** tab.

  When this option is selected and you launch the test harness in debug mode, the debugger suspends execution at the first executable line in the main method of the test harness, and Designer switches to the Debug perspective. The Debug view will show the test harness class and be positioned at the first statement in the main method. Also, the test harness will be opened in the Java editor, and it will be positioned to the first executable line in the main method.

When a Java class is suspended, use the views in the Debug perspective to inspect the state of the Java class and the actions in the Debug view toolbar to resume the execution. For more information about using the debugger, see the *Eclipse Java Development User Guide*.

Debugging a Java Service while its Class Runs in Designer

Use the following procedure to debug a Java service by debugging the Java class in the Service Development Project.

Note:
As a secondary method of debugging a Java service, you can debug a Java service that is running in Integration Server. This method requires setup on the Integration Server to change the way

the server starts and that can affect the server's performance. For more information, see " About Debugging a Java Service while its Class Runs in Designer " on page 456.

To debug a Java service

1. Open the Java service you want to debug by double clicking the service in the Package Navigator view.

2. Optionally, set breakpoints in the primary method of the Java service.

   To do so, in the Java service editor double click in the left margin next to the executable lines of code where you want breakpoints. Designer displays the breakpoint-enabled icon ( ) in the margin. For more information, see ""How to Suspend Execution of a Java Class while Debugging" on page 462 and the *Eclipse Java Development User Guide*.

3. Generate the test harness. For instructions, see "Creating a Test Harness" on page 458.

4. Optionally, create a Java Application launch configuration. For instructions, see "Creating a Java Application Launch Configuration" on page 460"0.

   If you do not create a Java Application launch configuration, Designer will create one automatically when you perform the next step.

5. Open the test harness in the editor, or select it in the Project Explorer view or Package Explorer view, and then select **Run > Debug As > Java Application**.

   - If you have no launch configuration for this test harness, Designer creates one and uses it
   - If you have a single launch configuration for this test harness, Designer uses it.
   - If you have multiple launch configurations for this test harness, Designer prompts you to select the launch configuration you want to use.

6. If you did not update the test harness to provide Integration Server login credentials, the test harness prompts you for the credentials. Specify the user name/password to connect to Integration Server and click **OK**.

7. If the Java service being tested has declared input parameters, the test harness displays an **Enter Input for *serviceName*** dialog box to prompt you for input values. To specify inputs, do one of the following:

   - Type values into the **Values** column. For more information, see "Entering Input for a Service" on page 406.
   - To load the input values that match the structure of the service input signature from a file, click **Load** to locate and select the file containing the input values.
   - To load input values from a file and replace the service input signature with the structure and data types in the file, click **Load and Replace**.

For more information about loading inputs from a file, see "Loading Input Values" on page 420.

8. Optionally, click **Save Inputs** to save the input values that you have specified so that you can use them to load input values in the future. For more information, see "Saving Input Values" on page 420.

9. Click **OK** to start launch the Java class in debug mode.

   The debugger executes the Java class. If you have set breakpoints or used the **Stop in main** option, the debugger suspends execution where you specified. If execution is suspended, Designer switches to the Debug perspective. For more information, see "How to Suspend Execution of a Java Class while Debugging" on page 462.

10. If execution suspends, use the views in the Debug perspective to inspect the state of the Java service and the actions in the Debug view toolbar to resume the execution. For more information about using the debugger, see the *Eclipse Java Development User Guide*.

    When the execution ends, Designer displays the **Output for *serviceName*** window with the service results.

11. In the **Output for *serviceName*** window, optionally click **Save Inputs** to save the service results to a file.

    This might be useful if you are testing another service that takes the results of this service as input. When debugging the next service you can load the results as input to execute that service.

12. Click **OK** to close the **Output for *serviceName*** window.

Viewing Service Results from Debugging a Java Service

When the Java Development Tools (JDT) debugger completes execution of a Java class that was launched in debug mode, the test harness obtains the outputs from the Java class and displays them in the **Output for *serviceName*** window. Designer does not display the results in the Results view.

In the **Output for *serviceName*** window, you can click **Save Inputs** to save the service results so that you can load them as inputs when running or debugging another service. This might be useful if you are testing another service that takes the results of this service as input.

To view service results

- In the **Output for *serviceName*** window, you can click **Save Inputs** to save the service results so that you can load them as inputs when running or debugging another service. This might be useful if you are testing another service that takes the results of this service as input.

About Debugging a Java Service while it Runs in Integration Server

As an alternative to debugging a Java service by using a test harness and debugging the local Java class in the Service Development Project, you can set up your Integration Server so that you can debug a Java service remotely. That is, use the features of Designer to debug a service that is running in Integration Server.

To debug a Java service using this method, you need to:

- For Integration Server version 9.7 or later, update and run startDebugMode.bat. For more information, see "Setting Up Integration Server Version 9.7 or Later for Remotely Debugging a Java Service" on page 466.
- For Integration Server version 9.0, 9.5.x, or 9.6, update the Integration Server setenv.bat/sh file to enable a debug port. For more information, see "Setting Up Integration Server Version 9.0, 9.5.x, or 9.6 for Remotely Debugging a Java Service" on page 467.
- Create a Java project in Designer from an existing source, which is the IS package in Integration Server. For more information, see "Creating a Java Project for an IS Package in Designer " on page 468.
- Create a Remote Java Application launch configuration to use when debugging the Java service. For more information, see "Creating a Remote Java Application Launch Configuration" on page 469.
- Open the Java service in Designer to set breakpoints and then execute it in debug mode.

Important:
Never remotely debug a Java service that is running on your production Integration Server. You should always use a development system.

Benefits of Debugging Java Services Running in Integration Server

The benefit of debugging a Java service in this way is that because the service is running in Integration Server, you can mimic the production environment more closely. There might be times when you cannot reproduce an issue when debugging using the Java class in the Service Development Project. In this situation, you can attempt to remotely debug the Java service.

Drawbacks of Debugging Java Services Running in Integration Server

The drawbacks of debugging a Java service that is running in Integration Server are:

- It requires more manual setup both in Integration Server and in Designer.
- If Integration Server and Designer run on different machines:

- You must logically map a drive from Designer to the Integration Server. For more information, see "Creating a Java Project for an IS Package in Designer " on page 468.
- If you use a firewall, you might need to open a firewall port to use while debugging. For more information, see either "Setting Up Integration Server Version 9.7 or Later for Remotely Debugging a Java Service" on page 466 or "Setting Up Integration Server Version 9.0, 9.5.x, or 9.6 for Remotely Debugging a Java Service" on page 467.

- If you set breakpoints in a Java service, the execution suspends every time you execute the Java service, whether you are debugging it from Designer, running it from Integration Server Administrator, or running it from an IS client.
- Running Integration Server with the debug port enabled degrades the server's performance.

Setting Up Integration Server Version 9.7 or Later for Remotely Debugging a Java Service

When using Designer to debug a Java service that is running in Integration Server version 9.7, you can set the port number and run the startDebugMode.bat/sh file to connect to a debug port.

To configure Integration Server version 9.7 or later for remotely debugging a Java service

1. Shut down Integration Server.
2. If you need to change the port number, perform the following:
   a. Open the startDebugMode.bat/sh file in a text editor. You can find the startDebugMode.bat/sh file in the following location:

      Software AG_directory \profiles\IS_*instance_name*\bin

   b. Locate and change the DEBUG_PORT property to specify the port on which the server should listen for the debugger to attach. The default is 10033.
   c. Save your changes and close the startDebugMode.bat/sh file.
3. If Integration Server and the debugging tool are on different machines and you require a firewall port, open a firewall port for the debug port.
4. Run startDebugMode.bat/sh.

   Integration Server logs the following on your console:

   ```
   "Debug enabled (portNumber)"
   Listening for transport dt_socket at address: portNumber
   ```

   Integration Server restarts.

Setting Up Integration Server Version 9.0, 9.5.x, or 9.6 for Remotely Debugging a Java Service

When using Designer to debug a Java service that is running in Integration Server version 9.0, 9.5.x, or 9.6 and later, make sure the Integration Server debug port is enabled at start up. You enable the debug port by updating the Integration Server setenv.bat (on Windows) or setenv.sh (on UNIX/Linux) file.

Important:*Before* performing the following procedure, make a backup copy of your setenv.bat/sh file.

To set up Integration Server 9.0, 9.5.x, or 9.6 for remotely debugging a Java service

1. Shut down Integration Server.
2. Open the .bat/sh file in a text editor. You can find the setenv.bat/sh file in the following location:

   On versions 9.0 to 9.5.x:

   IntegrationServer_directory\bin

   On version 9.6:

   IntegrationServer_directory\instances*instance_name*\bin
3. Locate and change the DEBUG_ENABLED property to `true`.
4. If you want to change the port number, locate and change the DEBUG_PORT property. The default is 9191.
5. Save your changes and close the setenv.bat/sh file.
6. If Integration Server and Designer are on different machines, if required, open a firewall port for the debug port.
7. If you are running Integration Server as a service, you must update the service for the changes in the setenv.bat file to take effect.

   To update the service, open a command window, navigate to the following location:

   On versions 9.0 to 9.5.x:

   Integration Server_directory \support\win32

   On version 9.6:

   Integration Server_directory \instances*instance_name*\support\win32
8. Run this command:

```
installSvc.bat update
```

9. Restart Integration Server.

   Integration Server logs the following on your console:

```
"Debug enabled (portNumber)"
Listening for transport dt_socket at address: portNumber
```

Creating a Java Project for an IS Package in Designer

In Designer, you need to create a Java project for the IS package that contains the Java service that you want to debug. This allows you to edit the Java service from Designer to set breakpoints. You set breakpoints to suspend the execution of the Java service during debugging, allowing you to inspect the state of the Java service in the middle of execution.

Note:
When opening a remote Java service in Designer, only add breakpoints; do not make other edits. To edit a Java service, follow the procedure described in "Editing an Existing Java Service" on page 316.

When creating the Java project in Designer, you create it from an existing source, which is the IS package on Integration Server.

If your Integration Server runs on a different machine than Designer, *before* performing the following procedure, map a logical drive from the machine on which Designer is running to the Integration Server machine that contains the IS package. To map a drive from Windows Explorer, use **Tools > Map Network Drive**. The mapped logical drive should be a shared drive that allows you to access the IS package. You can find IS packages in the following directory on the Integration Server machine:

IntegrationServer_directory\packages

To create a Java project for an IS package

1. Ensure Designer is using the Java perspective. If not, switch to it by selecting **Window > Open Perspective > Java**.

2. Select **File > New > Java Project**.

3. In the **Project name** field, type the name of the IS package that contains the Java service you want to debug.

4. Clear the **Use default location** check box.

5. Click **Browse** adjacent to the **Location** field. Navigate to and select the IS package that contains the Java service you want to debug. You can locate the IS packages in the following directory:

   Integration Server_directory \packages

Note:
If your Integration Server runs on a different machine than Designer, to access the directory containing the IS package when you click **Browse**, use the logical drive that you mapped before starting this procedure.

6. Click **Next**.

7. In the New Java Project window, select the **Libraries** tab.

8. Click **Add External Jars** and add the following jar files:

   - *Integration Server_directory* \lib\wm-isserver.jar
   - *Software AG_directory* \common\lib\wm-isclient.jar

9. Click **Finish**.

Creating a Remote Java Application Launch Configuration

To remotely debug a Java service on Integration Server from Designer, you need a Remote Java Application launch configuration that identifies the Integration Server debug port. After you create the launch configuration, you can then use it when executing the Java service from Designer, accessing Integration Server via the debug port.

When you execute the Java service in this way, if you have breakpoints set in the service, when execution suspends, you can use the Debug perspective to inspect the state of the Java service in the middle of execution.

Use the following procedure to create a Remote Java Application launch configuration.

To create a Remote Java Application launch configuration

1. In Designer: **Run > Debug Configurations**

2. In the Debug Configurations dialog box, select **Remote Java Application** and click to launch configuration.

3. In the right panel in the **Name** field, type a name for the launch configuration.

4. On the **Connect** tab in the **Project** field, specify the Java project you created for IS package that contains the Java service you want to debug.

5. In the **Connection Type** field, select **Standard (Socket Attach)**.

6. In the **Host** field, specify the name of remote Integration Server.

7. In the **Port** field, specify the port number of the Integration Server debug port that you defined.

8. Click **Apply**.

9. Click **Close**.

Debugging a Java Service while it Runs in Integration Server

Before you can remotely debug a Java service, complete the following setup:

- For Integration Server version 9.7 and later, modify and run the startDebugMode.bat/sh file. For more information, see "Setting Up Integration Server Version 9.7 or Later for Remotely Debugging a Java Service" on page 466.
- For Integration Server version 9.6, 9.5.x, and 9.0, update the setenv.bat/sh file. For more information, see "Setting Up Integration Server Version 9.0, 9.5.x, or 9.6 for Remotely Debugging a Java Service" on page 467.
- Create a Java project in Designer for the IS package containing the Java service you want to debug. For more information, see "Creating a Java Project for an IS Package in Designer " on page 468.
- Create a Remote Java Application launch configuration to use when remotely debugging the Java service. For more information, see "Creating a Remote Java Application Launch Configuration" on page 469.

After the setup is complete, you can debug the Java service. To do so, open the remote Java service to set breakpoints. Then run the Remote Java Application launch configuration, which you created earlier, in debug mode and execute the Java service. The debug session suspends execution at any breakpoints you set in any of the Java services in the Java project identified in the launch configuration. In Designer you can use the Debug perspective to inspect the state of the service execution.

To debug a Java service while it runs in Integration Server

1. Set breakpoints in the remote Java service by performing the following:

   a. From the Package Explorer view, expand the Java project you created for the IS package that contains the Java service you want to debug.

   b. Expand the code/source folder to locate the Java service you want to debug.

   c. Double click the Java service's .java file to open the service in the Java editor.

   d. To set breakpoints, double click in the left margin next to the executable lines of code where you want breakpoints. Designer displays the breakpoint enabled icon ( ) in the margin. For more information, see the *Eclipse Java Development User Guide*.

   Important:

After setting breakpoints, service execution will be suspended every time the service is executed. That is whether it is executed from Designer, Integration Server Administrator, or from an IS client.

2. Establish a listener that waits for the Java service to execute by running the launch configuration in debug mode.

   a. In Designer: **Run > Debug Configurations**.

   b. In the Debug Configurations dialog box, under **Remote Java Application** select the launch configuration you created for debugging the Java service.

   c. In the right panel, click **Debug**.

3. Execute the service in any way you want. For example, you can:

   - In Designer in the Package Explorer view, select the Java service and then select **Run As > Runs Service**.
   - Debug a flow service that invokes the Java service. While stepping through the flow service using the flow service debugger, when the step invokes the Java service executes, control is transferred to the Remote Java Application debugger.
   - Invoke the service from an IS client.

4. Switch to Debug perspective by selecting **Window > Open Perspective > Debug**.

Integration Server suspends the execution where you specified breakpoints. In Designer you can use the Debug perspective to inspect the state of the Java service. Use the actions in the Debug view toolbar to resume the execution. For more information about using the views in the Debug perspective, see the *Eclipse Java Development User Guide*.

24 Working with REST

- Approaches for Developing REST Resources 474
- Generating a REST Resource Using the Legacy Approach 474
- About the Folder for REST Resources Created Using the Legacy Approach 476
- Configuring a REST Resource Using the URL Template-Based Approach 476

REST (Representational State Transfer) is an architectural style that requires web applications to support HTTP methods such as GET, POST, PUT, PATCH, and DELETE, and to use a consistent, application-independent interface. Integration Server can act as a REST server or a REST client.

When Integration Server acts as a REST client, Integration Server sends specifically formatted requests to the REST server.

For Integration Server to act as a REST server, it must host services that perform the GET, PUT, POST, PATCH, and DELETE functions. These services perform functions that are specific to your application.

Note:
For more information about working with REST services in Integration Server, see the *REST Developer's Guide*.

Approaches for Developing REST Resources

Designer provides the following two different approaches for developing REST resources:

- Legacy approach, in which creating a REST resource includes creating the resource folder and flow services that correspond to supported HTTP methods.
- URL template-based approach, in which a URL template serves as a template for client requests to invoke a REST V2 resource.

The following sections explain the approaches in greater detail.

- "Generating a REST Resource Using the Legacy Approach" on page 474
- "Configuring a REST Resource Using the URL Template-Based Approach" on page 476

Generating a REST Resource Using the Legacy Approach

You can use the legacy approach to create a REST resource, including the REST resource folder, the flow services that correspond to HTTP methods, and, if selected, the _default flow service. When Designer generates the REST resource folder and services, Designer assigns each service the required names and creates an input signature that accepts the predefined parameters for a REST request.

REST resources generated using the legacy approach are invoked with the `rest` directive.

Creating a REST Resource Using the Legacy Approach

Consider the following when using Designer to create a REST resource:

- The REST resource name must be unique across the entire Integration Server namespace. That is, the fully qualified name of the REST resource must be unique on the Integration Server.
- You can select the HTTP methods for which you want Designer to create a service.

- You can instruct Designer to create a _default service to handle the HTTP methods for which there is not a specific service.
- The format of the request URI depends on the location of the REST resource. For more information about the format of the request URI, see the *REST Developer's Guide*.

To create a REST resource using the legacy approach

1. In the Service Development perspective, select **File > New > REST Resource**.
2. In the New REST Resource dialog box, select **REST Resource** as the type of REST resource to create.
3. Select the package or folder in which you want to create the REST resource.
4. In the **REST resource name** field, type a name for the REST resource using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
5. Click **Next**.
6. Under **HTTP methods**, select the check box next to the methods for which you want to create a separate service. Select the **Default** check box to create a service that handles any methods that are not selected.

   Note:
   You can select **Default** only if at least one of the HTTP methods is not selected.
7. Click **Finish**.

   Designer creates the REST resource folder and generates services for the selected HTTP methods.

Notes:

- The signatures for the flow services generated by Designer contain only the input parameters *$resourceID* and *$path* which are required by Designer. These parameter are optional and can be deleted.

  You might delete *$resourceID* and/or *$path* if your REST application does not use the values supplied by those variables.
- The _default service also contains a required input parameter named *$httpMethod*.
- After creating the services, you need to modify the service signature to include any additional input parameters expected by the service and any output parameters produced by the service.
- The services generated by Designer are empty. You must add processing logic to the generated services.

About the Folder for REST Resources Created Using the Legacy Approach

Designer uses the icon to indicate that the folder corresponds to a REST resource created using the legacy approach. Prior to Designer version 9.10, Designer used a regular folder icon for a REST resource folder. From version 9.10 onwards, Designer uses the icon for a REST resource.

When you change the contents of a folder, Designer analyzes the contents of each existing folder. If the folder contains any child service considered to be a REST resource, specifically _get, _put, _post, _patch, _delete, or _default, Designer changes the regular folder icon to the REST resource folder icon. However, if a folder contains a _delete service and other services used with OData, specifically _retrieve, _update, and _insert, Designer does not change the folder icon to be that of a REST resource folder icon.

Note:
If you upgrade to Integration Server version 9.10 or later from a version of Integration Server prior to 9.10, Designer uses the specified logic to convert regular folders to REST resource folders.

If the contents of a REST resource folder change such that Designer no longer considers the folder and its contents to be a REST resource, Designer replaces the REST resource folder icon with the regular folder icon. For example, suppose that a REST resource folder named topics contains services named _get, _delete, and _default. If you delete all of the services from topics, Designer uses the regular folder icon for the topics folder.

Configuring a REST Resource Using the URL Template-Based Approach

You can use the URL template-based approach to configure REST resources. In this approach, you define a URL template that serves as a template for client requests to use and invoke the resources.

REST resources configured using this approach, also known as *REST V2 resources*, are invoked with the `restv2` directive. For each resource, you must define operations that include the following:

- The format of the URL that REST clients must follow when sending requests to Integration Server acting as the REST server.
- The HTTP methods supported by the REST resource.
- The flow service associated with a resource operation. You can either associate an existing service with a resource operation or create a new service and associate it with the resource operation.

The URL template-based approach provides you with greater flexibility than the legacy approach in defining REST resources. For a REST V2 resource, you can define multiple operations and associate each operation with a URL template, HTTP methods, and a flow service. In addition, you can edit these details based on your requirements.

Note:
A REST V2 resource is represented by the icon in the Package Navigator view.

Creating a REST V2 Resource

To build a REST V2 resource, first create an empty REST V2 resource and then define operations for the resource.

To create a REST V2 Resource

1. In the Service Development perspective, select **File > New > REST Resource**.
2. On the New REST Resource dialog box, select **REST V2 Resource (Recommended)** as the type of REST resource to create.
3. Specify a name for the REST V2 resource and select the package or folder in which you want to create the resource.
4. Click **Finish**.

   Designer adds an empty REST V2 resource to the selected location in the namespace. For the created resource, you must define resource operations.

Defining a REST V2 Resource Operation

To define an operation of a REST V2 resource, you must specify the following:

- The format of the URL that REST clients must follow in their requests to Integration Server as the REST server.
- The supported HTTP methods.
- The associated service (Flow, Java, C, or Adapter service).

Important:
An operation in a REST V2 resource should be unique, that is, combination of a URL template and the supported HTTP methods must be unique).

You can associate either a new service (Flow, Java, C, or Adapter service) or an existing service (Flow, Java, C, or Adapter service) to a resource operation. Consider the following while associating a service (Flow, Java, C, or Adapter service) to a REST V2 resource operation:

- If you want to associate an existing flow service to a REST V2 resource operation, you must ensure that the dynamic parameter that you specify in the URL template already exists as a variable of type String in the input signature of the particular service. The service that you select need not reside in the same folder as the REST V2 resource.

  For information about static and dynamic parameters in a REST V2 resource, see *REST Developer's Guide*.
- If you rename a flow service associated with REST V2 resource operations and update its references, the impacted resource operations also are updated with the new name of the service.

- If you delete the flow service associated with a REST V2 resource operation or if you delete the input variable that is also referenced in the dynamic parameter of the URL template, the operation will not work when invoked by any subsequent client request to Integration Server.

To define a REST V2 resource operation

1. In the Package Navigator view of Designer, click the REST V2 resource for which you want to define the operation.
2. On the REST V2 Resource Configuration tab, click **Add**.
3. In the Add REST Resource Operation dialog box, specify the following:

Field	Description
REST URL	The URL template that a request from a client application must follow.
Service Name	The name of the flow service to associate with the REST V2 resource operation. You can specify a service name in either of the following ways: ■ Browse and select an existing service (Flow, Java, C, or Adapter service). ■ Specify the fully qualified name of an existing service (Flow, Java, C, or Adapter service). ■ Specify the name of a new flow service (Flow, Java, C, or Adapter service).
HTTP Methods	Select one or more HTTP methods for the REST V2 resource operation. **Note:** For a resource operation, you can select only from the HTTP methods that are supported for the associated service. For information about configuring the supported HTTP methods for an Integration Server service, see "Run Time Properties for Services" on page 1023.

4. Click **File > Save**.

   Integration Server validates the details specified for defining the new REST V2 resource operation. If the validation is successful, the new resource operation is added to the REST V2 Resource Configuration tab. Otherwise, Designer shows details of the validation errors, which you must fix before attempting to save the resource operation details again.

   Notes:

- If you specify a new service name for associating with the REST V2 resource operation, Integration Server creates a service (Flow, Java, C, or Adapter service) and adds it to the same folder as the REST V2 resource when saving the resource operation. In addition, the input signature of the new service includes the dynamic parameter specified in **REST URL**.
- You can edit the details of an existing REST V2 resource operation by selecting it on the REST V2 Resource Operation tab and clicking **Edit**.

Example of Defining a REST V2 Resource Operation

Consider a discussion application that maintains a database of discussions about different topics. If you want REST clients to interact with this application through Integration Server and read information about discussions related to a specific topic, under the folder called `app` create a REST V2 element named discussionNode with the resource name as `discussion` and define a resource operation that supports the HTTP GET method by providing the following details:

- **REST URL**: `/discussion/topic/{id}`
- **HTTP Methods**: `GET`

If you select an existing Integration Server service named `communication` to implement the specified resource operation, ensure that the service has a String input variable which corresponds to the dynamic parameter provided in the REST URL (in this case, `id`).

Based on the information specified, any client request to the REST server must be in the format: `GET /restv2/app:discussionNode/discussion/topic/{id}`, where `/restv2/app:discussionNode` is the template prefix used to invoke the `discussion` resource, and `{id}` is a dynamic parameter that accepts any value associated with a specific topic.

Deleting a REST V2 Resource Operation

Deleting a REST V2 resource operation also removes the specified URL template, the corresponding HTTP methods, and the associated flow service mapping. Therefore, ensure that any subsequent client request to Integration Server does not invoke the deleted REST resource operation because such a request will fail.

To delete a REST V2 resource operation

1. In the Package Navigator view of Designer, click the REST V2 resource for which you want to delete the operation.
2. On the REST V2 Resource Configuration tab, select an operation and click **Remove**.
3. Click **File > Save**.

   Integration Server deletes the selected resource operation.

25 Working with REST API Descriptors

- Overview of Creating a REST API Descriptor 482
- Creating a REST API Descriptor from a REST Resources 483
- Editing General Information for a REST API Descriptor 485
- Changing the Available MIME Types for a REST API Descriptor 486
- Working with REST Resources in a REST API Descriptor 486
- Viewing REST V2 Resources in Group by Tags Mode 494
- Running a REST V2 Resource Operation from REST API Descriptor Editor 495
- About REST Definitions 495
- Viewing the Swagger Document for a REST API Descriptor 496
- Mapping Integration Server Data Types to Swagger Data Types 496
- Creating a REST API Descriptor from a Swagger Document 497
- Publishing REST API Descriptors to API Portal 500

A REST API descriptor provides a way of describing the operations provided by one or more REST resources together with information about how to access those operations, the MIME types the resources consume and produce, and the expected input and output for the operations. Fundamentally, a REST API descriptor is composed of REST resources and information about how to access those resources. Using this information, Integration Server creates and maintains a Swagger document for the REST API descriptor. Integration Server generates the Swagger document based on version 2.0 of the Swagger specification.

You can create a REST API descriptor from REST resources created using either the legacy approach (invoked using the `rest` directive) or the URL template-based approach (REST V2 resources invoked using the `restv2` directive). You can also create a REST API descriptor containing REST V2 resources from a Swagger document.

For information about the approaches for configuring REST resources, see "Approaches for Developing REST Resources" on page 474.

Notes:

- Documentation for the REST API descriptor assumes back knowledge of REST concepts and the Swagger specification version 2.0.
- The information provided in this chapter applies to both the types of REST resources, that is, resources created using the legacy approach and those created using the URL template-based approach, unless explicitly specified for a particular type.

Overview of Creating a REST API Descriptor

Creating a REST API descriptor consists of the following general steps.

Stage 1	Creating REST resources. During this stage, you create the REST resources that you want to expose in the REST API descriptor. For more information about creating REST resources, see "Working with REST" on page 473.
Stage 2	Create a REST API descriptor on Integration Server. During this stage, you create the REST API descriptor, specifying information such as supported MIME types, protocols, and host:port. Additionally, you add REST resources to the REST API descriptor. Integration Server uses this information in the Swagger document that it dynamically generates for the REST API descriptor.
Stage 3	Modify information for the REST resources within the REST API descriptor. During this stage, you add or remove REST resources to the REST API descriptor.
Stage 4	Modify the operations for the REST resources in the REST API descriptor.

During this stage, you can change the MIME types consumed or produced by a specific operation. You can also review the source values assigned to parameters and add or remove operation responses.

Another way of creating a REST API descriptor is from a Swagger document. For details, see "Creating a REST API Descriptor from a Swagger Document" on page 497.

Creating a REST API Descriptor from a REST Resources

When you create a REST API descriptor, you select the REST resources that you want to include. You can also specify the MIME types that can be consumed and produced by the operations in the REST API descriptor, the supported protocols, and the base path for the REST API descriptor.

Keep the following in mind when creating a REST API descriptor:

- A REST resource must already exist on Integration Server.
- A REST API descriptor can include either REST resources created using the legacy approach or resources created using the URL template-based approach. The descriptor *cannot* include a combination of REST resources created using both the approaches.
- A REST API descriptor does not support duplicate variable names in the input parameters or output parameters. The REST API descriptor uses only the first variable with a particular name and ignores subsequent identically named variables. For example, if the input parameters include a String variable and a String List variable named "myField, the REST API descriptor uses only the first occurring "myField" in the REST definitions. However, if the output parameters also include a String variable named "myField", the REST API descriptor includes the output "myField" in the REST definitions.
- A REST API descriptor created from REST resources using the legacy approach can include a resource with a _default service. However, Integration Server does not create an operation that corresponds to the _default service.
- The names of documents and document references in a REST API descriptor must be unique. That is, a REST API descriptor does not support identically named documents and document references. If you create a REST API descriptor that contains more than one document or document reference with the same name, the generated Swagger document contains only one definition that corresponds to that document name.

To create a REST API descriptor

1. In the Service Development perspective of Designer, select **File > New > REST API Descriptor**.
2. In the Create a New REST API Descriptor dialog box, select the folder in which you want to save the REST API descriptor.
3. In the **Element** name field, type a name for the REST API descriptor using any combination of letters, numbers, and the underscore character. For more information about restricted characters, see "Guidelines for Naming Elements" on page 42.

4. Click **Next**.

5. Depending on the type of REST resources you want to include in the REST API descriptor, select either **REST V2 Resources (Recommended)** or **REST Resources**.

> **Note:**
> If you want to create a REST API descriptor from a Swagger document, see "Creating a REST API Descriptor from a Swagger Document" on page 497.

6. In the Specify REST API Descriptor General Details panel, provide the following information:

In this field...	Specify...
Title	A title for the application represented by the REST API descriptor.
Description	A description of the application represented by the REST API descriptor.
Application Version	A version number for the application. The default is 1.0. **Note:** The **Application Version** is not the version of the Swagger specification.
Host:Port Name	The host and port for the Integration Server on which the application resides in the format: *host*:*port* By default, the REST API descriptor uses the primary host:port of the Integration Server to which Designer is connected.
Path	The base path for the REST API descriptor. The path must begin with a "/" (slash). For REST resources using the legacy approach, Integration Server obtains this value from the watt.server.RESTDirective server configuration parameter and the default path is `/rest`. For REST resources using the URL template-based approach, the default value of path is `/rad/<namespace name of REST API descriptor>`
Consumes	Select the MIME types that operations in the REST API descriptor can consume. The MIME types you select here apply to all the operations in the REST API descriptor. However, you can override the MIME types on a per operation basis.

In this field...	Specify...
	For more information, see "Changing the MIME Types for an Operation in a REST Resource" on page 489. If you do not see the MIME type you want to use listed, you can update the list of available MIME types. For more information, see "Changing the Available MIME Types for a REST API Descriptor" on page 486.
Produces	Select the MIME types that operations in the REST API descriptor can produce. The MIME types you select here apply to all the operations in the REST API descriptor. However, you can override the MIME types on a per operation basis. For more information, see "Changing the MIME Types for an Operation in a REST Resource" on page 489. If you do not see the MIME type you want to use listed, you can update the list of available MIME types. For more information, see "Changing the Available MIME Types for a REST API Descriptor" on page 486.

7. Click **Next**.

8. In the Select the REST Resources panel, select one or more REST resources to include in the REST API descriptor.

9. Click **Finish**.

   Designer creates the REST API descriptor using the information you provided along with the selected REST resources.

Editing General Information for a REST API Descriptor

When you create a REST API descriptor, Integration Server uses the general information that you supplied such as title, path, version, and MIME types to populate the General tab in the REST API descriptor editor. You can change this information at any time. Integration Server updates the Swagger document displayed on the Swagger tab dynamically.

To edit the general information for a REST API descriptor

1. In the Package Navigator view of Designer, open and lock the REST API descriptor for which you want to edit general information.

2. On the General tab, edit the general information for the REST API descriptor based on your requirements.

3. Click **File > Save**.

Changing the Available MIME Types for a REST API Descriptor

The MIME types preference determines the list of available MIME types for a REST API descriptor. The MIME types specified in the preference determine the contents of the **Consumes** and **Produces** fields in the REST API Descriptor wizard, on the General tab in the REST API descriptor editor, and for an operation on the REST Resources tab.

Note:

You can access the MIME types preference by clicking the in the upper right corner of the General tab in the REST API descriptor editor.

To change the available MIME types

1. In Designer, click **Window > Preferences**.
2. In the Preferences dialog box, select **Software AG > Service Development > REST API Descriptor**.
3. Under MIME types, do one of the following:

To...	Do this...
Add a MIME type	Click **Add**. In the Add new MIME type dialog box, enter the MIME type and click **OK**.
Edit a MIME type	Select the MIME type that you want to edit and click **Edit**. In the Edit MIME type dialog box, modify the selected MIME type and click **OK**.
Delete a MIME type	Select the MIME type that you want to delete and click **Remove**.
Restore the default MIME types	Click **Restore Defaults**.

4. Click **Apply** to save your changes to the list of available MIME types.
5. Click **OK**.

Working with REST Resources in a REST API Descriptor

After you create a REST API descriptor, you can add more REST resources to the descriptor, remove REST resource from the descriptor, set a path or suffix for the resource, and work with the operations in the descriptor.

Adding REST Resources to a REST API Descriptor

You can include additional REST resources in any REST API descriptor. Keep the following information in mind when selecting REST resources to add to a REST API descriptor:

- A REST API descriptor can include either REST resources created using the legacy approach or resources created using the URL template-based approach (REST V2 resources). The descriptor *cannot* include a combination of REST resources created using both the approaches.
- A REST API descriptor created from REST resources using the legacy approach can include a resource with a _default service. However, Integration Server does not create an operation that corresponds to the _default service.
- A REST API descriptor does not support duplicate variable names in the input parameters or output parameters. The REST API descriptor uses only the first variable with a particular name and ignores subsequent identically named variables. Note that a variable in the input parameters can have the same name as a variable in the output parameters.

To add a REST resource to a REST API descriptor

1. In the Package Navigator view, lock and open the REST API descriptor to which you want to add a REST resource.
2. Click the REST Resources tab.
3. On the REST API descriptor toolbar, right-click in the REST Resources tab and select **Add REST Resource**.
4. In the Select one or more REST resources to include in the REST API descriptor dialog box, select one or more REST resources to add to the REST API descriptor and click **OK**.

   Designer adds the REST resource to the REST API descriptor.
5. Click **File > Save**.

Removing REST Resources from a REST API Descriptor

You can remove REST resources from a REST API descriptor. However, a REST API descriptor must contain at least one REST resource. Designer prevents you from deleting the last REST resource in a REST API descriptor.

To remove a REST resource from a REST API descriptor

1. In the Package Navigator view, lock and open the REST API descriptor from which you want to remove a REST resource.
2. Click the REST Resources tab.

3. Select the REST resource that you want to remove.

4. Click ✖on the REST API descriptor editor toolbar or right-click the selected REST resource and select **Delete**.

   Designer removes the selected REST resource from the REST API descriptor.

5. Click **File > Save**.

Setting the Path or Suffix for a REST Resource

You can set the path or suffix used for a REST resource included in a REST API descriptor.

By default, each REST resource in a REST API descriptor derives its path from the namespace of the REST resource. For example, if the REST resource is named myREST.myRESTResource, the path is "/myREST.myRESTResource". However, you might not want to expose the namespace of the REST resource in the Swagger document. You can override the default path with a path that you specify. For example, you could use `/customers/premium` or `/myPath`.

By default, there is no suffix for the REST resources in a REST API descriptor. However, if you want users who invoke the REST resource to include query parameters, you can specify that information in the suffix. Integration Server appends the suffix to the path. For example, if you want the request to invoke a REST resource to include the $resourceID, specify a suffix of: `/{$resourceID}`. If you want the request URL to include the $resourceID and the $path, specify a suffix of `/{$resourceID}/{$path}`.

If you change the path, the suffix, or both, make sure that Integration Server can resolve the resulting resource path. The path must be invokable by Integration Server.

Note:
The values that you specify for the path and suffix apply only to the REST resource as it used in this REST API descriptor. It does not affect the same REST resource used in another REST API descriptor or the REST resource itself.

To set the path or suffix for a REST resource

1. In the Package Navigator view, lock and open the REST API descriptor for which you want to specify a path and/or suffix for a REST resource.

2. Click the REST Resources tab.

3. Select the REST resource for which you want to specify a path and/or suffix.

4. To specify a path for the REST resource, in the Properties view, next to **Path**, type the path that you want to use for the REST resource.

   If you do not include "/" as the first character in the **Path** property, Integration Server adds it in the Swagger document.

5. To specify a suffix for the REST resource, next to **Suffix**, type the suffix than you want to use for the REST resource.

6. Click **File > Save**.

Working with Operations

In a REST API descriptor, each included REST resource lists the associated operations. For each operation, Designer displays the parameters used by the operation and the responses returned by the operation. Designer also displays a description of the operation which Designer obtains from Comments tab for the corresponding resource operation. For each operation in a REST API descriptor, you can do the following:

- Change the MIME types that the operation consumes or produces
- Review the assigned source of the parameter and change the source if necessary.
- Add and remove responses.

Changing the MIME Types for an Operation in a REST Resource

When you create a REST API descriptor, you specify the MIME types that the APIs can consume and produce. Each REST resource in the REST API descriptor inherits these MIME types. However, for an individual operation in a REST resource included in a REST API descriptor, you can select different MIME types that the operations in the REST resource consume or produce.

Note:
The MIME types set for a REST resource operation in a descriptor override the MIME types set for the parent REST API descriptor. That is, the consumes and produces MIME types specified for an individual REST resource operation replace the MIME types specified for the REST API descriptor.

To change the MIME types for an operation in a REST resource

1. In the Package Navigator view, lock and open the REST API descriptor that contains the REST resource operation which you want to specify MIME types.

2. Click the REST Resources tab.

3. Select the REST resource that contains the operation for which you want to specify MIME types.

4. Expand the operation for which you want to specify MIME types.

5. To change the MIME types that the operation can consume, next to **Consumes**, select the MIME types that the operation can consume.

If no MIME types are selected, the operation can consume the MIME types listed on the General tab of the REST API descriptor.

If you do not see the MIME type you want to use listed, you can update the list of available MIME types. For more information, see "Changing the Available MIME Types for a REST API Descriptor" on page 486.

6. To change the MIME types that the operation can produce, next to **Produces**, select the MIME types that the operation con produce.

   If no MIME types are selected, the operation can produce the MIME types listed on the General tab of the REST API descriptor.

   If you do not see the MIME type you want to use listed, you can update the list of available MIME types. For more information, see "Changing the Available MIME Types for a REST API Descriptor" on page 486.

7. Click **File > Save**.

About the Operation Parameters

When you select the Parameters for an operation, Designer displays a list of the input parameters in the REST resource service that corresponds to the operation. For each parameter, Designer displays the following information:

- **Name**, which matches the name of the input parameter in the corresponding service.
- **Source**, which indicates the source of the parameters in the incoming request. The source corresponds to the "in" field for a fixed field in a parameter object as defined by the Swagger specification. A parameter can have one of the following source values:
  - QUERY
  - HEADER
  - PATH
  - FORMDATA
  - BODY

  Designer assigns the parameter a Source value. If an input parameter for a service is a document or document reference, Designer assigns the parameter a source of BODY. You cannot change the source of a parameter set to BODY. For an operation parameter of a REST V2 resource, you cannot change the source if set to BODY or PATH.
- **Type**, which specifies the data type of the parameter. For a mapping of Integration Server data types to Swagger data types, see "Mapping Integration Server Data Types to Swagger Data Types" on page 496.
- **Description**, which provides a description of the parameter. Designer obtains the description from the **Comments** property for the parameter in the corresponding REST resource service.

- **Required**, which indicates whether or not the parameter is required in the input. Designer obtains this value from the **Required** property for the parameter in the corresponding REST resource service.
- **Include in Swagger**, which you can select to indicate whether or not to include a parameter in the Swagger document for the REST API descriptor. If you want to exclude an already selected parameter from the Swagger document, clear the checkbox for the parameter under the **Include in Swagger** column.

  Note:
  This option is available only for parameters of a REST V2 resource operation.

You can change the name, description, or required values for a parameter by modifying the parameter in the corresponding REST resource service. However, you can only make the following changes to a parameter in a REST API descriptor:

- Change the source for a parameter.
- For a REST API descriptor containing REST V2 resource operations, specify whether to include a parameter in the Swagger definition.

Reviewing and Changing the Assigned Source for an Operation Parameter

Designer assigns each parameter a source. The source value for a parameter indicates how the input parameter is sent in a request. Designer usually assigns operation parameters a source of FORMDATA, except in the following situations:

- If an input parameter for a service associated with a resource operation is a document or a document reference, Designer assigns the parameter a source of BODY.
- For a REST V2 resource, if an input parameter for a service associated with the resource operation is referenced as a dynamic parameter in the URL format, Designer assigns the parameter a source of PATH. If the input parameter is referenced as a static parameter in the URL format, Designer assigns the parameter a source of HEADER.

When you add a REST resource to a REST API descriptor, you should verify that each parameter uses the source value that matches the service implementation exposed by the operation.

Important:
For a REST API descriptor containing resources created using the legacy approach, you cannot change the assigned source for a parameter if the source value is BODY. For a REST API descriptor containing REST V2 resources, you cannot change the assigned source for a parameter if the value is either BODY or PATH.

To review and change the source for a parameter

1. In the Package Navigator view, lock and open the REST API descriptor that contains the REST resource operation for which you want to change the parameter source.
2. Click the REST Resources tab.

3. Expand the REST resource for which you want to view and change the parameter source.

4. Expand the operation for which you want to view and/or change the assigned source.

5. Click Parameters.

   Designer displays the parameters for the operation.

6. To change the source for a parameter, click the cell for the parameter in the **Source** column and select the source you want to assign to the parameter.

7. Repeat step 6 for each parameter in the operation.

8. Click **File > Save**.

About Operation Responses

An operation response is a possible response a client can expect when invoking an operation in the REST API descriptor. When you add a REST resource to a REST API descriptor, Designer creates two responses for each operation in the REST resources:

- Successful with a returned HTTP status code of 200.
- Access Denied with a returned HTTP status code of 401.

You can add or delete operation responses. For example, you might want to have a response for successful execution of an operation and a separate response for each known error the operation can return. Designer requires that an operation have at least one response. If an operation has only one response, it is a best practice to make sure that the response is for successful execution of the operation.

When an operation returns output, the returned output needs to have some explanation, such as the returned parameters. In general, Integration Server derives the output from the output signature of a service corresponding to the REST operation. If a service has output parameters, Integration Server generates a REST definition that corresponds to the output signature.

When an operation returns output with the response and the operation has output parameters, the HTTP status code includes a hyperlink. When clicked, Designer opens the REST Definitions tab and displays the response document for the operation.

Adding an Operation Response

You can add a response to any operation in a REST resource that is included in a REST API descriptor.

To add a response to an operation

1. In the Package Navigator view, lock and open the REST API descriptor that contains the REST resource operation for which you want to add a response.

2. Click the REST Resources tab.

3. Expand the REST resource for which you want to add an operation response.

4. Expand the operation for which you want to add the operation response.

5. Click Responses.

   Designer displays the responses for the operation.

6. Click .

7. In the Add Response dialog box, supply the following information:

In this field...	Specify...
Code	The HTTP status code that the operation can return
Description	Short description of the response.
Return Output	Whether or not the operation returns output with the response. Select one of the following: ■ **True** if the operation returns output with the response. Typically a REST operation returns output for successful HTTP status code, such as status code of 200. ■ **False** if the operation does not return output with the response. This is the default.

8. Click **File > Save**.

Removing an Operation Response

You can remove an operation from a response, including any operations that Designer adds by default. However, each operation must have at least one response. If an operation has only one response, it is a best practice to make sure that the response is for successful execution of the operation.

To remove a response from an operation

1. In the Package Navigator view, lock and open the REST API descriptor that contains the REST resource operation for which you want to add a response.

2. Click the REST Resources tab.

3. Expand the REST resource for which you want to add an operation response.

4. Expand the operation for which you want to add the operation response.

5. Click Responses.

   Designer displays the responses for the operation.

6. Click ✖.

   Designer deletes the operation response. Designer disables ✖ if the operation contains only one response.

7. Click **File > Save**.

Viewing REST V2 Resources in Group by Tags Mode

The REST Resources tab for a REST API descriptor displays the REST resources grouped by the resource name. If a REST API descriptor contains many REST resources, the REST Resources tab may appear cluttered. To provide a more organized view of the resources in a REST API descriptor, you can group the resources by tags.

Operations in a REST API descriptor are associated with a tag. When you view the REST Resources in a group by tags mode, Designer organizes all the operations (part of multiple REST Resources) under the tags associated with each operation. An operation can have multiple tags associated with it. In case of multiple tags, the operation is grouped under the first defined tag.

Any operation that does not have a tag associated with it, is grouped under the `default` tag.

To view REST Resources in Group by Tags mode

1. In the Package Navigator view, open the REST API descriptor for which you want to group the resources using the Group by Tags method.

2. Click the REST Resources tab.

3. Click on the top right corner of REST Resources tab.

   Designer displays the REST V2 resources in the Group by Tags mode.

Running a REST V2 Resource Operation from REST API Descriptor Editor

Use Designer to execute individual operations of a REST V2 resource. You can create launch configurations to run a REST V2 resource operation. It is similar to creating launch configurations to run services. For details, see "Running Services" on page 403.

You can use one of the following ways to execute an operation:

- Using the Try it out option in the Swagger UI: Designer provides a Swagger UI view that surfaces the open source Swagger UI and Try it out option developed for Swagger. For more information about these open source tools, see the Swagger website.
- Using the Run option in the REST API descriptor editor: For details about using the editor to execute an operation, see "Executing an Operation Using Editor" on page 495.

Executing an Operation Using Editor

You can use the **Run** option in REST API descriptor editor to execute an operation.

To run a REST V2 resource operation

1. In the Package Navigator view, open the REST API descriptor which contains the operation that you want to execute .
2. Click the REST Resources tab.
3. Select the operation that you want to execute.
4. On the top right corner of REST Resources tab, click .
5. In the **Enter Input for *<REST URL>*** dialog box, specify input values for the operation.
6. Click **OK**.

   Designer runs the operation and displays the results in the Results view.

About REST Definitions

The REST Definitions tab for a REST API descriptor provides a consolidated view of the parameters used by the operations for the REST resources included in the REST API descriptor. The contents of the REST Definitions tab become the definitions object in the Swagger document.

Integration Server creates a REST definition for each of the following:

- Input parameter that is a Document, Document List, Document Reference, or Document Reference List.

- An input signature that includes the above along with one or more scalar parameters. The REST definition includes the scalar parameters and a reference to the REST definition created for the Document, Document List, Document Reference, or Document Reference.
- Output parameters returned by an operation.

The REST Definitions tab is read only. For information about REST Definitions properties, see "REST Definition Properties" on page 1005 and "REST Definition Parameter Properties" on page 1006.

Viewing the Swagger Document for a REST API Descriptor

Integration Server dynamically generates a Swagger document for the REST API descriptor whenever you make or save updates to the REST API descriptor or to the REST resources that correspond to operations in the descriptor. Designer displays the Swagger document on the Swagger tab.

To view the Swagger document for a REST API descriptor

1. In the Package Navigator view, open the REST API descriptor for which you want to view the Swagger document.
2. Click the Swagger tab.

   The Swagger document is display only.

Mapping Integration Server Data Types to Swagger Data Types

Integration Server maps IS data types to Swagger data types as follows:

IS Data Type	Swagger Data Types	Swagger Data Format
String	string	
String List	string	
String Table	string	
Object	string	binary
Object List	string	binary
Document	ref	
Document List	ref	
Document Reference	ref	
Document Reference List	ref	
java.lang.Byte	string	byte

IS Data Type	Swagger Data Types	Swagger Data Format
java.lang.Boolean	boolean	
java.util.Date	string	date
byte[]	string	byte
java.lang.integer	integer	int32
java.lang.Short	integer	int32
java.lang.Long	integer	int64
java.lang.Double	number	double
java.lang.Character	string	
java.math.BigDecimal	number	double
java.math.BigInteger	integer	int64

Note:
For any parameter that is an array or of type String table, for the corresponding parameter in the Swagger document, Integration Server sets the “type” as array.

Creating a REST API Descriptor from a Swagger Document

You can create REST API descriptors from external Swagger documents accessible through a URL, a REST resource on API Portal, or a REST resource residing in CentraSite. Keep the following in mind when creating a REST API descriptor from a Swagger document:

- To create a REST API descriptor from a REST resource in CentraSite or API Portal, Designer must be configured to connect to CentraSite or API Portal.
- The Swagger document that you use to create a REST API descriptor must be based on the Swagger Specification version 2.0.
- Integration Server does not validate the contents of the external Swagger document and generates the assets based on the Swagger document used. Therefore, ensure that you validate the Swagger document before creating a REST API descriptor.
- Designer supports only the following Swagger file formats: JSON and YAML.
- Integration Server does not support the following specification parameters in a Swagger document:
  - `security`
  - `securityDefinitions`
  - `externalDocs`

To create a REST API descriptor from a Swagger document

1. In the Service Development perspective of Designer, select **File > New > REST API Descriptor**.

2. In the Create a New REST API Descriptor dialog box, select the folder in which you want to save the REST API descriptor.

3. In the **Element** name field, type a name for the REST API descriptor using any combination of letters, numbers, and the underscore character. For more information about restricted characters, see "Guidelines for Naming Elements" on page 42.

4. Click **Next**.

5. In the Select the Source Type dialog box, click **Swagger Document** and click **Next**.

6. In the Select the Swagger Document dialog box, select one of the following:

Select...	To...
API Portal	Generate a REST API descriptor from APIs present in the API Portal.
CentraSite	Generate a REST API descriptor from a REST resource available in CentraSite.
File/URL	Generate a REST API descriptor from a Swagger document that resides on the file system or on the Internet.
Import Swagger based on Tags	Group the operations defined in the Swagger document based on the tags. If you do not select this option, Designer groups the operations by path.

7. If you specified **API Portal** as the source, click **Next** and then under Select the API from API Portal list, select the API that you want to use to create the REST API descriptor.

8. If you specified **CentraSite** as the source, click **Next** and then under Select REST Resource from CentraSite, select the REST resource in CentraSite that you want to use to create the REST API descriptor.

9. If you specified **File/URL** as the source, do one of the following:

   - Enter the URL for the Swagger document. The URL should begin with http:// or https://.

     Provide your username and password to use the Swagger document from the website.

- Click **Browse** to navigate to and select a Swagger document on your local file system.

10. Click **Finish**.

    Designer creates the REST API descriptor. Designer also creates the associated services (**services**), definitions (**docTypes**), and resources (**resources**) and places them under a folder that has the same name as the REST API descriptor appended with an underscore (_). The Properties view displays the source URL and source URI value for the REST API descriptor and its associated document type. For more information about the fields in the Properties view, see "REST API Descriptor Properties" on page 1003.

    Note:
    You cannot edit the elements, the general information in the General tab, or the MIME type of the REST API descriptor that you created. You can only edit the base path of the REST API descriptor and the service implementation of the services generated.

    Note:
    Do not edit, rename, or delete the **services**, **docTypes**, and **resources** folders, its subfolders, or the parent folder containing them all. Also, when you move the REST API descriptor to a different location in Designer, ensure that you move the corresponding folder containing the **services**, **docTypes**, and **resources** folders.

Refreshing a REST API Descriptor

If the Swagger document used to create a REST API descriptor changes, you may want to refresh the REST API descriptor to reflect the recent changes or to refresh the REST API descriptor using a different Swagger document. For example, if you created a REST API descriptor from a Swagger document that has since changed to include, edit, or remove an peration, path, or definition, you can refresh the REST API descriptor. Refreshing the REST API descriptor does the following:

- Updates the REST API descriptor, its associated services, document types, or resources to reflect changes in existing elements in the updated Swagger document.
- Adds elements, such as services, document types, or resources to the REST API descriptor to reflect new elements in the updated Swagger document.
- Removes REST API descriptor elements that correspond to elements that have been removed from the updated Swagger document.
- Preserves any changes you made to the service implementation in the REST API descriptor since it was created from the original Swagger document.

Note:
In Designer, the option to refresh a REST API descriptor is available only for those REST API descriptors that are created using a Swagger document.

To refresh a REST API Descriptor

1. In the Package Navigator view, right click the Rest API descriptor that you want to refresh.

2. Click **Refresh REST API Descriptor**.

3. Review the informational message about potential changes to the existing REST API descriptor and click **OK** to continue with refresh.

4. If the **Source URI** property specifies a location for the original Swagger document, if there is a change in the Swagger document location, specify the new location of the file for refreshing the REST API descriptor. Click **Yes** to the Swagger document for which the location has changed or click **No** to use the file at the specified location as the source.

   Designer refreshes the REST API descriptor. If Designer cannot refresh a REST API descriptor, Designer rolls back to the last saved version of the REST API descriptor. If refresh is not successful, use the messages returned by Designer and Integration Server logs to determine the issue.

Publishing REST API Descriptors to API Portal

You can publish REST API descriptors created on Integration Server to webMethods API Portal. API Portal is a web-based portal that allows organizations to securely expose different types of APIs to external users. You must use Designer to connect to API Portal, and publish the REST API descriptors.

You can select one or more REST API descriptors in Designer for publishing to API Portal. You can update the details of an already published descriptor on API Portal. However, you cannot retract an already published descriptor. Instead, you can delete the published descriptors from API Portal, if required.

Before you can publish REST API descriptors to API Portal, you must define a connection configuration in Designer. For information about working with connection configurations for API Portal, see "Connecting to webMethods API Portal for Publishing REST API Descriptors" on page 929.

To publish a REST API descriptor

1. In the Package Navigator view of Designer, right-click the REST API descriptor you want to publish and select **Publish**.

   If you want to publish multiple REST API descriptors in a folder or a package, right-click the particular folder or package and select **Publish**.

   Note:
   If you want to update details of a REST API descriptor that is already published to API Portal, you can select the particular descriptor for publishing. In such a situation, Integration Server overwrites the existing details for that descriptor on API Portal.

2. In the Asset Publish dialog box, select **API Portal** as the destination for publishing the descriptors and click **Next**.

The Publish Assets toAPI Portal dialog box lists the API descriptor you selected in Step 1. If you selected a folder or a package, all the descriptors in the particular folder or package are selected in the dialog box.

3. In the Publish Assets to API Portaldialog box, refine your selection of API descriptors by selecting or clearing the appropriate check boxes, and click **Finish**.

   The publish process starts for the REST API descriptors that you selected. Designer displays the results of the publish process for the selected descriptors on the **Published Metadata** screen.

   Note:
   For information about the errors that occur during the publish process, see the Integration Server server log.

26 Working with OData Services

- Understanding OData Service Terminology 504
- Supported and Unsupported OData Features 505
- Overview of Creating an OData Service 506
- Creating an OData Service 506
- Adding OData Elements to the OData Service 509
- Adding Properties to the OData Elements 510
- Adding Associations to OData Elements 511
- Editing the OData Service 511
- Synchronizing the External Entity Type 512
- How Integration Server Processes an OData Service Request 512
- Querying Data Using $filter 513

OData (Open Data Protocol) enables applications to expose data or resources as a data service that clients can access within corporate networks and across the Internet. It provides a REST-based protocol for performing create, read, update and delete (CRUD) operations against resources that are exposed as data services.

Integration Server acts as an OData service provider and supports OData version 2.0.

You can use the Service Development perspective in Designer to create *OData services*. An OData service can be described as an endpoint service that is based on the OData protocol and allows access to data. The OData service exposes an OData entity data model that contains data organized and described in a standard manner.

The OData service consists of a set of entity types, external entity types, and complex types, their properties, and the associations between the entity types. When you create an OData service, Integration Server, acting as an OData service provider, generates the required OData service implementations as flow services that can perform the CRUD operations for each entity type.

These flow services will:

- Be located in a folder whose fully qualified name is unique across the entire Integration Server namespace.
- Be named _insert, _retrieve, _update, and _delete to perform the create, read, update, and delete operations respectively.
- Accept certain predefined input parameters that are passed in through the OData request.

Note:
While the services _retrieve, _update, _insert, and _delete might appear to be regular flow services, it is the naming convention and location of the services that instruct Integration Server to treat them as OData service implementations.

Understanding OData Service Terminology

Before creating an OData service, you may find it helpful to first understand the following terminology related to OData support in Integration Server:

- **OData Service.** Service that is based on the OData protocol and exposes an OData entity data model that contains data organized and described in a standard manner.
- **OData Elements.** Represents entity types and complex types in an OData service.
- **Entity Type.** Entity types are named structured types with a key. For example, uniquely identifiable records such as a `Customer` or `Employee`.
- **External Entity Type.** External entity types are entity types available through an external source provider. For example, if you select to use webMethods Adapter for JDBC as your external source provider or source type, you can use Database Tables as entity types to create your entities.

- **Complex Type.** Structural types consisting of a list of properties but with no key. For example, `Address`, which includes city, street, state, and country. You can access a complex type only when they are added as a complex property to an entity type.
- **Properties.** Used to define the characteristics of OData elements. For example, a `Customer` entity type may have properties such as `CustomerId`, `Name`, and `Address`. Properties can be simple or complex. Simple property can contain primitive types (such as a string, an integer, or a Boolean value). Complex property can contain structured data such as a complex type.
- **Associations.** Represents the relationship between two entity types. For example, relationship between `Customer` and `Order`. An association has two ends and each end specifies the entity type attached to that end. Associations can be **Single** (unidirectional) or **Bidirectional** depending on the number of entity types that can be at that end of the association.
- **Navigation Property.** Represents the association end and allows navigation from an entity to related entities. For example, `Product` can have a navigation property to `Category` and `Category` can, in return, have a navigation link to one or more products.

Supported and Unsupported OData Features

The OData service implementation in Integration Server supports the following OData features:

- ATOM, JSON, and XML formats for representing the resources that are exposed as data services.
- CRUD operations for each entity type.
- System query options such as `$select`, `$filter`, `$orderby`, `$top`, `$skip`, `$format`, `$inlinecount`, and `$count`.
- Association and navigation properties.

The OData service implementation in Integration Server does NOT support the following actions:

- Rename, copy, or move OData services.
- Copy and paste OData elements, such as entity types and complex types, within and outside the OData service editor.
- Rename OData associations and navigation properties.
- Rename External Entity Types.

The following OData features are NOT supported:

- Function Imports
- Referential constraints
- Annotations
- Collection type data type
- Lambda query operators

Overview of Creating an OData Service

Creating an OData service is a process that involves the following basic stages.

Stage 1 Create a new OData service on Integration Server.

During this stage, you use the Service Development perspective to create a new OData service. You can choose to create an empty OData service or use an external source type.

Stage 2 Add OData elements to the OData service in the OData service editor.

During this stage, you specify the OData elements, namely **Entity Type**, **External Entity Type**, and **Complex Type** that define the entity model that this OData service exposes.

Stage 3 Specify properties for the OData elements.

During this stage, you specify the properties that define the structure and characteristics of entity types and complex types. You also set the properties for each simple and complex property in the Properties view.

Stage 4 Define the associations between the entity types.

During this stage, you specify the associations that represents the relationship between two entity types. You also set the properties for each association in the Properties view.

Upon defining the associations, Designer creates the OData navigation elements under the corresponding entity types.

Stage 5 Save the OData service.

During this stage, you save the OData service after ensuring that the OData service you have created contains all the elements, properties, and associations required to expose an OData entity data model.

Stage 6 Add processing logic to the generated services.

During this stage, you add processing logic to the OData service implementations that Integration Server generates upon saving the OData service.

Creating an OData Service

You use the Service Development perspective in Designer to create an OData service and specify the entity types, external entity types, complex types, properties, and associations.

To create an OData service

1. In the Service Development perspective, select **File > New > OData Service**.

2. In the New OData Service dialog box, select the folder in which you want to create the OData service.

3. In the **Element name** field, type a name for the OData service using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "Working with Elements" on page 39.

4. Click **Finish**.

   Alternatively, you can also click **Next**.

   a. In the Select a Source Type screen, select **Empty OData Service**

   b. Click **Finish**.

   Designer creates the OData service and displays the details in the OData service editor. You must now add OData elements, specify their properties, and define the association between the entity types.

   - To add OData elements, see "Adding OData Elements to the OData Service" on page 509.
   - To specify the properties of OData elements, see "Adding Properties to the OData Elements" on page 510.
   - To define associations between OData elements, see "Adding Associations to OData Elements" on page 511.

5. Click **File > Save** to save the OData service.

   Designer creates a folder with the same name as the OData service, with a prefix of an "_" (underscore). Inside this folder, Designer creates folders for each entity type. These folders contain OData service implementations that are named _insert, _retrieve, _update, and _delete can perform the create, read, update, and delete operations for each entity.

   The signatures for the OData service implementations generated by Designer contain document variables generated using the entity types and its properties. You must not modify the signature of the generated OData service implementations. You must only add processing logic to these services.

Creating an OData Service Using an External Source Type

You use the Service Development perspective in Designer to create an OData service using an external source type and specify the entity types, external entity types, complex types, properties, and associations.

Before you create an OData Service using an external source, ensure that the selected OData enabled external source type is configured to Integration Server with a valid connection. For example, if you choose to use webMethods Adapter for JDBC as your external source provider, ensure that you create a connection in Integration Server for Adapter for JDBC.

To create an OData service using an External Source Type

1. In the Service Development perspective, select **File > New > OData Service**.
2. In the New OData Service dialog box, select the folder in which you want to create the OData service.
3. In the **Element name** field, type a name for the OData service using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "Working with Elements" on page 39
4. Click **Next**.
5. In the Select The Source Type screen, select **External Source Type**.

   List of the available external source types appear.
6. Select the external source that you want to use as the external source type for the OData service. Click **Next**.
7. In the Select a Connection Alias screen, select the connection to use with the source type from the **Connection Name** list and click **Next**.

   Designer retrieves the entity details from the database and displays it in the Select the Entity and Properties screen.
8. In the Select the Entity and Properties screen, select the entities that you want to include in your OData service. You can also change the Java Type for the entity selected.

   Note:
   If you go back to the previous screens and return to the Select the Entity and Properties screen, the entity list does not get refreshed. To get the new list of entities, start creating the OData service again.
9. Click **Finish**.

   Designer creates the OData service and displays the details in the OData service editor. You can now add additional OData elements, specify properties, and define the association between the entity types.

   - To add OData elements, see "Adding OData Elements to the OData Service" on page 509.
   - To specify the properties of OData elements, see "Adding Properties to the OData Elements" on page 510.
   - To define associations between OData elements, see "Adding Associations to OData Elements" on page 511.

10. Click **File > Save** to save the OData service.

    Designer creates a folder with the same name as the OData service, with a prefix of an "_" (underscore). Inside this folder, Designer creates folders for each entity type. These folders contain OData service implementations that are named _insert, _retrieve, _update, and _delete can perform the create, read, update, and delete operations for each entity.

    The signatures for the OData service implementations generated by Designer contain document variables generated using the entity types and its properties. You must not modify the signature of the generated OData service implementations. Also, Integration Server adds the processing logic to these services.

Adding OData Elements to the OData Service

Entity types are named structured types with a key. For example, uniquely identifiable records such as a `Customer` or `Employee`. Complex types are structural types consisting of a list of properties but with no key. For example, `Address`, which includes city, street, state, and country. External entity types are entity types available through an external source provider. For example, if you choose to use webMethods Adapter for JDBC as your external source provider or source type, you can use Database Tables as entity types to create your entities.

To add OData elements to an OData service

1. Open the OData service to which you want to add an OData element.

2. In the Palette view of the OData editor, under **OData Elements** perform one of the following:, select the type of element you want to add and drag it to the Tree tab of the OData service.

   For details about adding each type of element, see the following steps.

   Note:
   If the Palette view is not visible, display it by clicking on the right side of the editor.

3. To add an entity type as an OData element, do the following:

   a. Select **Entity Type** and drag it to the Tree tab.

   b. Provide a name for the entity type.

      Note:
      The name of an entity type must be unique among all the entity types in the OData service.

4. To add the external entity type as an OData element, do the following:

   a. Select **External Entity Type** and drag it to the Tree tab.

b. Select an external source from the list and click **Next**

c. In the Select a Connection Alias dialog box, select the connection to use with the source type from the **Connection Name** list and click and click **Next**.

d. In the Select the Entity and Properties dialog box, select the entities that you want to include in your OData service. You can also change the Java Type for the entity selected.

e. Click **Finish**.

The selected external entity types appear on the Tree tab of the OData service.

5. To add a complex type as an OData element, do the following:

a. Select **Complex Type** and drag it to the Tree tab.

b. Provide a name for the complex type.

> **Note:**
> The name of the complex type element must be unique across all of the complex types in the OData service.

6. Click **File > Save** to save the OData elements.

Adding Properties to the OData Elements

Entity types and complex types contain properties that define their characteristics. For example, a `Customer` entity type may have properties such as `CustomerId`, `Name`, and `Address`. Simple property can contain primitive types (such as a string, an integer, or a Boolean value). Complex property can contain structured data such as a complex type.

To add properties

1. Open the OData service and select the entity type or complex type to which you want to add property.

2. In the Palette view of the OData editor, under **Properties**, select **Simple** and/or **Complex** properties and drag it to the Tree tab to add simple and complex properties to the OData service.

> **Note:**
>
> If the Palette view is not visible, display it by clicking on the right side of the editor.

The entity type and complex type elements can have one or more properties. Each entity type should have at least one property that is a key.

Note:
You cannot add properties to external entity type.

3. Click **File > Save** to save the properties.

Adding Associations to OData Elements

You can specify an association to represent the relationship between two entity types (such as `Customer` and `Order`). Each association has two ends and each end of the association must specify the entity type attached to that end. Associations can be **Single** or **Bidirectional** depending on the number of entity types that can be at the end of the association.

To add associations

1. Open the OData service and select the entity type, external entity type, or complex type to which you want to add association.

2. In the Palette view of the OData editor, under **Associations**, select **Single** or **Bidirectional** and drag it to the Tree tab.

   Note:
   If the Palette view is not visible, display it by clicking on the right side of the editor.

3. In the Entity Association dialog box, select the principal and dependent entity types for the ends of the association.

4. Click **OK**.

   Designer creates the associations and displays the OData navigation elements under the corresponding entity types. In case of unidirectional association, a navigation element is added only to the specific entity type that is the association end. In case of bidirectional association, navigation elements are added to both entity types that are at the association ends.

5. Click **File > Save** to save the association.

Editing the OData Service

You can use the OData service editor to modify an OData service.

Keep the following points in mind when editing an OData service:

- Do not modify the signature of the generated OData service implementations. You must only add processing logic to these services.
- To modify the signature of any of the generated OData service implementations, you must modify the entities and their associations in the OData service.

To edit an OData Service

1. In the Package Navigator view, navigate to and open the OData service that you want to edit. The OData service opens in the OData service editor.

2. Make the necessary modifications to the OData elements, properties, or associations. You can also modify the corresponding properties in the Properties view, if required.

3. Click **File > Save**.

4. Refresh the OData service to update the corresponding OData service implementations by right-clicking the OData service in the Package Navigator view and selecting **Refresh**.

Synchronizing the External Entity Type

You can use the OData Sync feature to synchronize the properties of a selected external entity type with the latest changes made at the external source provider level. You can also use the Sync feature to edit the properties of an external entity type.

To sync and edit an external entity type

1. In the Package Navigator view, navigate to and open the OData service which contains the external entity that you want to edit. The OData service opens in the OData service editor.

2. Select the external entity that you want to edit and click the Sync icon on the Designer toolbar or right-click and select **Sync**.

3. In the External Entity Sync dialog box, you can make the following modifications to the selected external entity:

   - Select new properties available for the entity.
   - Remove the existing properties.
   - Change the data type for the **Java Type**.
   - Click the key icon to make or remove a property as primary key.

4. When you finish making changes to the external entity, click **Sync**.

   The updated the external entity is displayed in the OData service editor.

How Integration Server Processes an OData Service Request

Integration Server uses Apache Olingo 2.0.1 libraries to process OData service requests. To do this, Integration Server uses the `odata` directive. A directive is a way to access or invoke resources.

You can specify the odata directive as follows:

```
http://host:port/odata/parent_context/resource
```

Here, *parent_context* is the OData service node in the Integration Server namespace and *resource* refers to the name of an entity type or a collection of instances of an entity type.

For example:

```
http://localhost:5555/odata/container:company/Products
```

Note:
When processing an OData service request, Integration Server checks the user name associated with the request against the appropriate access control list (ACL) associated with the service. If the user belongs to a group that is listed in the ACL, the server accepts the request. Otherwise the server rejects the request. Ensure that the OData service has the required ACLs associated with it so that Integration Server processed the requests.

Querying Data Using $filter

You can use filter expressions in OData requests to filter and return only those results that match the expressions specified. You do this by adding the `$filter` system query option to the end of the OData request.

You can specify the `$filter` query option in an OData request as follows:

```
http://host:port/odata/parent_context/resource?$filter=expressions
```

Here, *parent_context* is the OData service node in the Integration Server namespace, *resource* refers to the name of an entity type or a collection of instances of an entity type, and *expressions* are the filter expressions.

Integration Server supports the following logical operators:

Operator	Description	Example	Requests...
`eq`	Equals	`http://localhost:5555/odata/OData.svc/Customers?$filter=Name eq 'Smith'`	All customers whose name is 'Smith'.
`ne`	Not equal	`http://localhost:5555/odata/OData.svc/Customers?$filter=Name ne 'Smith'`	All customers whose name is not 'Smith'.
`gt`	Greater than	`http://localhost:5555/odata/OData.svc/Products?$filter=Price gt 1000`	All products with price greater than 1000.
`ge`	Greater than or equal	`http://localhost:5555/odata/OData.svc/Products?$filter=Price ge 1000`	All products with price greater than or equal to 1000.

Operator	Description	Example	Requests...
`lt`	Less than	`http://localhost:5555/odata/OData.svc/Products?$filter=Price lt 1000`	All products with price less than 1000.
`le`	Less than or equal	`http://localhost:5555/odata/OData.svc/Products?$filter=Price le 1000`	All products with price less than or equal to 1000.
`and`	Logical and	`http://localhost:5555/odata/OData.svc/Products?$filter=Category eq 'laptop' and Price lt 1000`	All products of category laptop and price less than 1000.
`or`	Logical or	`http://localhost:5555/odata/OData.svc/Customers?$filter=City eq 'Paris' or City eq 'London'`	All customers from the city of Paris or London.

Note:
You can also use custom filters instead of OData built-in filters while using the $filter system query option. To use your custom filters, set the **Use custom filter** property of the OData service to `True`. You can then specify custom filter queries as the value for the *$filter* parameter of the _retrieve and _update OData service implementations.

27 Working with Document Types

- Creating an IS Document Type 516
- Editing Document Types 551
- About Universal Names and Document Types 552
- Printing an IS Document Type 552
- Working with Publishable Document Types 553
- Deleting Publishable Document Types 574
- About Testing Publishable Document Types 575
- About Synchronizing Publishable Document Types 580
- Publishing Documents as JMS Messages 589

An IS document type contains a set of fields used to define the structure and type of data in a document (IData object). You can use an IS document type to specify input or output parameters for a service or specification. You can also use an IS document type to build a document or document list field and as the blueprint for pipeline validation and document (IData object) validation.

IS document types can provide the following benefits:

- Using an IS document type as the input or output signature for a service can reduce the effort required to build a flow.
- Using an IS document type to build document or document list fields can reduce the effort needed to declare input or output parameters or the effort/time needed to build other document fields.
- IS document types improve accuracy, because there is less opportunity to introduce a typing error typing field names.

IS document types make future changes easier to implement, because you can make a change in one place (the IS document type) rather than everywhere the IS document type is used.

Creating an IS Document Type

You can create an IS document type in the following ways:

- Create an empty IS document type and define the structure of the document type yourself by inserting fields.
- Create an IS document type from a source file, such as an XML Schema, DTD, XML document, an event type, JSON object, or e-form template. The structure and content of the IS document type will match that of the source file.
- Create an IS document type from a Broker document type.
- Create an IS document type from a flat file schema.

Creating an Empty IS Document Type

When you create an empty IS document type, you insert fields to define the contents and structure of the IS document type.

To create an empty IS document type

1. In the Service Development perspective, select **File > New > Document Type**
2. In the New Document Type dialog box, select the folder in which you want to save the IS document type.
3. In the **Element name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

4. Click **Next**.

5. On the Select the Source Type panel, select **None**.

6. Click **Finish** to create the empty IS document type.

Adding Fields to an IS Document Type

By adding fields to an IS document type, you define the structure and content of instances of the IS document type.

Note:
When defining an IS document type, avoid adding identically named fields to the IS document. In particular, avoid adding identically named fields that are the same data type. While Designer allows this, the identically named fields may cause some anomalies especially with regards to mapping data in the pipeline.

Keep the following points in mind if you intend to make the IS document type publishable:

- If you intend to use Broker as the messaging provider, keep in mind that the Broker has restrictions for field names. When a document is published to Broker, fields with names that do not meet these restrictions will be passed through byBroker. If you create a trigger that subscribes to the publishable document type, any filters that include field names containing restricted characters will be saved on the Integration Server only. The filters will not be saved on the Broker, possibly affecting Integration Server performance. For more information, see "Creating a webMethods Messaging Trigger " on page 682.
- If you intend to use Universal Messaging as the messaging provider and use protocol buffers as the encoding type, keep in mind that some field names might not work with protocol buffers. If a publishable document type contains fields that use unsupported characters, these fields and their contents will be passed through to Universal Messaging. Subscribing triggers will decode the field properly. However, Universal Messaging cannot filter on the contents of these fields.

  Note:
  If a publishable document types uses protocol buffers encoding type, Software AG recommends using a letter as the first character of a field name and avoiding special characters with the exception of the _ (underscore) character.

To add fields to the IS document type

1. In Package Navigator view, double-click the document type to which you want to add fields.

   The document type opens in the **Tree** tab of the document type editor.

2. Drag the document type field that you want to define from the Palette to the **Tree** tab in the editor.

3. Type the name of the field and then press ENTER.

   Note: Designer prevents the insertion of fields named *_env* in an IS document type. For details about the *_env* field, see "About the Envelope Field" on page 560.

4. With the field selected, set field properties and apply constraints in the Properties view (optional).

5. If the field is a document or a document list, repeat the preceding steps to define and set the properties and constraints for each of its members. Use to indent each member field beneath the document or document list field.

6. Enter comments or notes, if any, in the **Comments** tab.

7. Select **File > Save**.

Note: Designer displays small symbols next to a field icon to indicate validation constraints. Designer uses to indicate an optional field. Designer uses the ‡ symbol to denote a field with a content constraint. For information about applying constraints to fields, see "About Variable Constraints" on page 607.

Creating an IS Document Type from an XML Document, DTD, or XML Schema

You can create an IS document type based on the structure and content of a source file, such as an XML Schema definition, DTD, or XML document. Keep the following points in mind when creating an IS document type from a source file.

- When you base the IS document type on an XML Schema definition or a DTD, Integration Server creates an IS document type *and* an IS schema. The IS document type has the same structure and field constraints as the source document. The IS schema contains the elements, attributes, and data types defined in the XML Schema or DTD. The IS document type, which displays the fields and structure of the source document, uses links to the IS schema to obtain content type information about named simple types.
- When creating a field from an attribute declaration, Integration Server inserts the @ symbol at the beginning of the field name. For example, an attribute named `myAttribute` in the source file corresponds to a field named *@myAttribute* in the IS document type.
- If an IS document type was created from a source, Designer displays the location of the source in the **Source URI** property. Designer also sets the **Linked to source** property to true which prevents any editing of the document type contents. To edit the document type contents, you first need to make the document type editable by breaking the link to the source. For information about allowing editing of elements derived from a source, see "Allowing Editing of Derived Elements" on page 47
- Integration Server does not create IS document types or IS schemas from an XML schema definition (XSD) if the XSD contains a type definition derived by extension and that type

definition contains a direct or indirect reference to itself. If Integration Server encounters a type definition that contains a recursive extension while creating an IS document type or an IS schema from an XSD, Integration Server throws a StackOverflowError and does not continue creating the IS document type or IS schema.

Creating an IS Document Type from an XML Document

You can create an IS document type that matches the structure and fields in an XML document.

To create an IS document type from an XML document

1. In the Service Development perspective, select **File > New > Document Type**
2. In the New Document Type dialog box, select the folder in which you want to save the IS document type.
3. In the **Element Name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
4. Click **Next**.
5. On the Select the Source Type panel, select **XML**. Click **Next**.
6. On the Select a Source Location panel, under **Source location**, do one of the following in the **File/URL** field:
   - To create the IS document type from an XML document that resides on the Internet, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
   - To create the IS document type from an XML document on your local file system, type in the path and file name, or click the **Browse** button to navigate to and select the file.
7. Click **Finish** to create the IS document type.

   If you want to add or edit fields in the IS document type, see "Creating an Empty IS Document Type" on page 516.

Creating an IS Document Type from a DTD

When creating an IS document type from a DTD, keep in mind that Integration Server assumes that the DTD is UTF8-encoded. If the DTD is not UTF8-encoded, add the XML prolog to the top of the DTD and explicitly state the encoding. For example, for a DTD encoded in 8859-1, you would insert the following at the top of the document:

```
<?xml version="1.0" encoding="8859-1" ?>
```

To create an IS document type from a DTD

1. In the Service Development perspective, select **File > New > Document Type**

2. In the New Document Type dialog box, select the folder in which you want to save the IS document type.

3. In the **Element name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

4. Click **Next**.

5. On the Select the Source Type panel, select **DTD**. Click **Next**.

6. On the Select a Source Location panel, under **Source location**, do one of the following next to **File/URL**:

   - To create the IS document type from an DTD that resides on the Internet, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
   - To create the IS document type from a DTD on your local file system, type in the path and file name, or click the **Browse** button to navigate to and select the file.

7. Click **Next**.

8. Under **Select the root node**, select the root element of the DTD.

9. Under **Element reference handling**, select one of the following:

   - Select **Only generate document types for elements with multiple references** to instruct Integration Server to create a separate document type for a referenced element only when the DTD contains multiple references to that element.

     If an element is referenced multiple times, Integration Server creates a separate document type for the element. Integration Server replaces each element reference with a document reference field.

     If an element is referenced only once, Integration Server defines the element in line by replacing the element reference with a document field.

   - Select **Always generate document types for referenced elements** to instruct Integration Server to always create a separate document type for a referenced element even if it is referenced only once. In the document type, Integration Server replaces each element reference with a document reference field.

10. Click **Finish**.

    Integration Server generates the IS document type and IS schema. Designer displays any errors or warnings that occur during document type generation.

Creating an IS Document Type from an XML Schema Definition

Keep the following points in mind when creating an IS document type from an XML Schema definition:

- You can specify whether Integration Server enforces strict, lax, or no content model compliance when generating the document type. Content models provide a formal description of the structure and allowed content for a complex type. The type of compliance that you specify can affect whether Integration Server generates an IS document type from a particular XML Schema definition successfully. Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type from any XML schema definition that contains those items.
- Integration Server can create separate IS document types for named complex types or expand documents inline within one document type. For more information, see "Determining How to Represent Complex Types in Document Types" on page 543.
- Integration Server can create one field for a substitution group or create fields for every member element in a substitution group. For more information, see "Generating Fields for Substitution Groups" on page 549.
- To create an IS document type from an existing event type, use the event type as the source. Make sure you have access to the Event Type Store.
- To create an IS document type from an XML Schema definition in CentraSite, Designer must be configured to connect to CentraSite.
- You can also create an IS document type from an XML Schema definition asset in CentraSite by dragging and dropping the schema asset from the Registry Explorer view into Package Navigator view.
- When creating an IS document type from an XML Schema definition that contains a large number of complex type definitions, and you want Integration Server to create a separate IS document for each complex type definition, you may need to increase the number of elements that Designer maintains in cache. If the cache is not large enough to include all of the generated IS document types, then Designer will have to repeatedly retrieve the document types from Integration Server while creating the IS document type. This increases network traffic and can prolong the time needed to generate the IS document type. If the cache is large enough to contain all of the IS document types and other elements generated by Designer while creating an IS document type, Designer might create the IS document type more quickly. To increase the number of elements cached by Designer, see "Caching Elements" on page 72.

To create an IS document type from an XML Schema definition

1. In the Service Development perspective, select **File > New > Document Type**
2. In the New Document Type dialog box, select the folder in which you want to save the IS document type.

3. In the **Element name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

   If you are creating an IS document type from an event type, you may want to use the event type name as the name for the IS document type.

4. Click **Next**.

5. On the Select a Source Type panel, select **XML Schema**. Click **Next**.

6. On the Select a Source Location panel, under **Source location**, do one of the following to specify the source file for the document type:

   - To use an XML schema definition in CentraSite as the source, select **CentraSite**.
   - To use an XML Schema definition that resides on the Internet as the source, select **File/URL**. Then, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
   - To use an XML Schema definition that resides on your local file system as the source, select **File/URL**. Then, type in the path and file name, or click the **Browse** button to navigate to and select the file.
   - To use an existing event type as the source, navigate to the Event Type Store and select the XML Schema definition for the event type.

     The default location of the Event Type Store is: *Software AG_directory* \common\EventTypeStore

7. Click **Next**.

8. If you selected CentraSite as the source, under **Select XML Schema from CentraSite**, select the XML Schema definition in CentraSite that you want to use to create the IS document type. Click **Next**.

   If Designer is not configured to connect to CentraSite, Designer displays the **CentraSite > Connections** preference page and prompts you to configure a connection to CentraSite.

9. On the Select Processing Options panel, under **Schema domain**, specify the schema domain to which any generated IS schemas will belong. Do one of the following:

   - To add the IS schema to the default schema domain, select **Use default schema domain**.
   - To add the IS schemas to a specified schema domain, select **Use specified schema domain** and provide the name of the schema domain in the text box. A valid schema domain name is any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

10. Under **Content model compliance**, select one of the following to indicate how strictly Integration Server represents content models from the XML Schema definition in the resulting IS document type.

Select...	To...
Strict	Generate the IS document type only if Integration Server can represent the content models defined in the XML Schema definition correctly. Document type generation fails if Integration Server cannot accurately represent the content models in the source XML Schema definition. Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type from any XML schema definition that contains those items.
Lax	When possible, generate an IS document type that correctly represents the content models for the complex types defined in the XML schema definition. If Integration Server cannot correctly represent the content model in the XML Schema definition in the resulting IS document type, Integration Server generates the IS document type using a compliance mode of None. When you select lax compliance, Integration Server will generate the IS document type even if the content models in the XML schema definition cannot be represented correctly.
None	Generate an IS document type that does not necessarily represent or maintain the content models in the source XML Schema definition. When compliance is set to none, Integration Server generates IS document types the same way they were generated in Integration Server releases prior to version 8.2.

11. If you selected strict or lax compliance, next to **Preserve text position**, do one of the following to specify whether document types generated from complex types that allow mixed content will contain multiple `*body` fields to preserve the location of text in instance documents.

    - Select the **Preserve text position** check box to indicate that the document type generated for a complex type that allows mixed content preserves the locations for text in instance documents. The resulting document type contains a **body* field after each field and includes a leading **body* field. In instance documents for this document type, Integration Server places text that appears after a field in the **body*.
    - Clear the **Preserve text position** check box to indicate that the document type generated for a complex type that allows mixed content does not preserve the locations for text in instance documents. The resulting document type contains a single **body* field at the top of the document type. In instance documents for this document type, text data around fields is all placed in the same **body* field.

12. If this document type will be used as the input or output signature of a service exposed as a web service and you want to enable streaming of MTOM attachment for elements of type base64Binary, select the **Enable MTOM streaming for elements of type base64Binary** check box.

    For more information about streaming of MTOM attachments, see the *Web Services Developer's Guide*

13. If you want Integration Server to use the Xerces Java parser to validate the XML Schema definition, select the **Validate schema using Xerces** check box.

    Note: Integration Server automatically uses an internal schema parser to validate the XML Schema definition. However, the Xerces Java parser provides stricter validation than the internal schema parser. As a result, some schemas that the internal schema parser considers to be valid might be considered invalid by the Xerces Java parser.

14. Click **Next**.

15. On the Select Root Node panel, under **Select the root node,** select the elements that you want to use as the root elements for the IS document type. The resulting IS document type will contain all of the selected root elements as top-level fields in the generated IS document type

    To select multiple elements, press the CTRL key while selecting elements.

    If you are creating an IS document type from an event type, select the element whose name is the event type name as the root node. For example, if the event type name is PartInventoryLow, select the PartInventoryLow element as the root element.

    If Integration Server determines that the XML Schema definition is invalid, the Select Root Node panel displays an error message to that effect. Click **Cancel** to abandon the attempt to create a document type.

16. Under **Element reference handling**, select one of the following to determine how Integration Server handles references to global elements of complex type:

Select...	To...
Only generate document types for elements with multiple references	Instruct Integration Server to create a separate document type for a referenced element only when the XML Schema definition contains multiple references to that element. If an element is referenced multiple times, Integration Server creates a separate document type for the element. Integration Server replaces each element reference with a document reference field. If an element is referenced only once, Integration Server defines the element in line by replacing the element reference with a document field.

Select...	To...
Always generate document types for referenced elements	Instruct Integration Server to always create a separate document type for a referenced element even if it is referenced only once. In the document type, Integration Server replaces each element reference with a document reference field

Note: Integration Server always replaces an element reference to an element declaration of simple type with an inline field of type String.

17. Under **Complex type handling**, select one of the following to indicate how Integration Server handles references to named complex type definitions:

Select...	To...
Expand complex types inline	Use a document field defined in line to represent the content of a referenced complex type definition.
Generate document types for complex types	Create a separate IS document type to represent the content for a referenced complex type definition. The resulting IS document type for the root element represents the element of complex type using a document reference field. In turn, this document reference field refers to the IS document type created for the complex type definition. Integration Server generates a separate IS document type for any types derived from the referenced complex types. For more information about derived types, see "Derived Types and IS Document Types" on page 545.

Note: Integration Server always represents an anonymous complex type using a document field defined inline.

18. If you selected **Generate document types for complex types** and you want to register each document type with the complex type definition from which it was created, select the **Register document type with schema type** check box.

Note:
If you want derived type support for document creation and validation, select the **Register document types with schema type** check box. For more information, see "Registering Document Types with Their Schema Types" on page 547.

19. If you want Integration Server to generate IS document types for all complex types in the XML Schema definition regardless of whether the types are referenced by elements or other type definitions, select the **Generate document types for all complex types in XML Schema** check box.

If you leave this check box cleared, Integration Server generates a separate IS document type for a complex type only if the complex type is referenced or is derived from a referenced complex type.

If you are creating an IS document type from an event type, clear the **Generate document types for all complex types in XML Schema** check box.

20. Click **Next**.

21. On the Assign Prefixes panel, if you want the IS document type to use different prefixes than those specified in the XML schema definition, select the prefix you want to change and enter a new prefix. Repeat this step for each namespace prefix that you want to change.

    Note:
    The prefix you assign must be unique and must be a valid XML NCName as defined by the specification http://www.w3.org/TR/REC-xml-names/#NT-NCName.

22. Click **Finish**. Integration Server generates the IS document type(s) and IS schema and saves it on the server. Designer displays them in the Package Navigator view.

Notes:

- Integration Server uses the internal schema parser to validate the XML schema definition. If you selected the **Validate schema using Xerces** check box, Integration Server also uses the Xerces Java parser to validate the XML Schema definition. With either parser, if the XML Schema does not conform syntactically to the schema for XML Schemas defined in *XML Schema Part 1: Structures*, Integration Server does not create an IS document type or an IS schema. Instead, Designer displays an error message that lists the number, title, location, and description of the validation errors within the XML Schema definition.

  Note: Integration Server uses Xerces Java parser version J-2.11.0. Limitations for this version are listed at http://xerces.apache.org/xerces2-j/xml-schema.html.

- When validating XML schema definitions, Integration Server uses the Perl5 regular expression compiler instead of the XML regular expression syntax defined by the World Wide Web Consortium for the XML Schema standard. As a result, in XML schema definitions consumed by Integration Server, the pattern constraining facet must use valid Perl regular expression syntax. If the supplied pattern does not use proper Perl regular expression syntax, Integration Server considers the pattern to be invalid.

  Note:
  If the watt.core.datatype.usejavaregex configuration parameter is set to true, Integration Server uses the Java regular expression compiler instead of the Perl5 regular expression compiler. When the parameter is true, the pattern constraining facet in XML schema definitions must use valid syntax as defined by the Java regular expression.

- If you selected strict compliance and Integration Server cannot represent the content model in the complex type accurately, Integration Server does not generate any IS document types.

- If you selected lax compliance and indicated that Integration Server should preserve text locations for content types that allow mixed content (you selected the **Preserve text position** check box), Integration Server adds **body* fields in the document type only if the complex type allows mixed content and Integration Server can correctly represent the content model declared in the complex type definition. If Integration Server cannot represent the content model in an IS document type, Integration Server adds a single **body* field to the document type.
- The contents of an IS document type with a **Model type** property value other than "Unordered" cannot be modified.
- If the XML schema definition contains an element reference to an element declaration whose type is a named complex type definition (as opposed to an anonymous complex type definition), Integration Server creates an IS document type for the named complex type definition. In the IS document type for the root element, Integration Server uses document reference field to represent the element reference. An exception to this behavior is the situation in which the element reference is the only reference to the complex type definition and the **Only generate document types for elements with multiple references** option is selected. In this situation, Integration Server uses document field defined in line to represent the content of the referenced complex type.
- Integration Server uses the prefixes declared in the XML Schema or the ones you specified as part of the field names. Field names have the format *prefix*:*elementName* or *prefix*:*@attributeName*.
- If the XML Schema does not use prefixes, the Integration Server creates prefixes for each unique namespace and uses those prefixes in the field names. Integration Server uses "ns" as the prefix for the first namespace, "ns1" for the second namespace, "ns2".
- If the XML Schema definition contains a user-specified namespace prefix and a default namespace declaration, both pointing to the same namespace URI, Integration Server uses the user-specified namespace prefix and not the default namespace.
- If the namespace prefix in the XML Schema as well as the default namespace point to the same namespace URI, Integration Server gives preference to the user-specified namespace prefix over the default namespace.
- When creating an IS document type from an XML Schema definition that imports multiple schemas from the same target namespace, Integration Server throws Xerces validation errors indicating that the element declaration, attribute declaration, or type definition cannot be found. The Xerces Java parser honors the first <import> and ignores the others. To work around this issue, you can do one of the following:
  - Combine the schemas from the same target namespace into a single XML Schema definition. Then change the XML schema definition to import the merged schema only.
  - When creating the IS document type clear the **Validate schema using Xerces** check box to disable schema validation by the Xerces Java parser. When generating the IS document type, Integration Server will not use the Xerces Java parser to validate the schemas associated with the XML Schema definition.

Creating IS Document Types from JSON Objects

You can create an IS document type based on the contents of a source JSON object.

When creating an IS document type from a JSON object, keep in mind that Designer displays the location of the source JSON object in the **Source URI** property. Designer also sets the **Linked to source** property to true which prevents any editing of the document type contents. To edit the document type contents, you first need to make the document type editable by breaking the link to the source. For information about allowing editing of elements derived from a source, see "Allowing Editing of Derived Elements" on page 47.

Mapping JSON Data Types

Integration Server maps IS data type from JSON data types as follows:

JSON data type	IS data type
object	Document
string	String
number (real)	Float or Double Java wrapper, depending on the option you select for mapping real numbers.
number (integer)	Integer or Long Java wrapper, depending on the option you select for mapping integers.
true/false	Boolean Java wrapper type
array	Array of an Integration Server data type. ■ If the JSON object contains a string array, Integration Server sets the corresponding field in the IS document type to String List. ■ If the JSON object contains a two-dimensional string array, Integration Server sets the corresponding field in the IS document type to String Table. ■ If the JSON object contains a array having different types of objects (example, integer, string, etc.), Integration Server sets the corresponding field in the IS document type to Object List with Java wrapper type as UNKNOWN. **Note:**Integration Server does not support the following multi-dimensional arrays in JSON objects: ■ Two-dimensional arrays of data types other than string

JSON data type	IS data type
	■ Multi-dimensional arrays of three or more dimensions (three-dimensional, four-dimensional, etc.) For IS document types created from a JSON object using Designer, these kinds of arrays become an Object with a Java wrapper type of UNKNOWN.
array of null	Object List
null	null
All others	String **Note:** If an object has a toString() implementation, Integration Server uses that implementation. If the object does not provide a toString() implementation, Integration Server uses Object.toString(). Object.toString() returns the class name and hexadecimal representation of the hash code of the object, such as "javax.namining.InitialContext@3ae6f00b".

Note:
If JSON text begins with an array at the root and the array is unnamed, when parsing the JSON text, Integration Server uses a fixed name of *$rootArray* for the array value. The *$rootArray* field appears in the pipeline. When creating a JSON response, if the pipeline contains *$rootArray* with an array value at its root, Integration Server discards the *$rootArray* name and transforms the array value into a JSON array.

Note:
With regards to using Integration Server data types with JSON, Integration Server supports only those types that can be mapped to a JSON value as defined in https://tools.ietf.org/html/rfc7159#section-3. Integration Servercan take any valid arbitrary JSON text and convert it to an IData. Integration Server must be able to convert the resulting IData to JSON text that is identical to the original text. If Integration Server cannot do that for an Integration Server data type, then Integration Server does not support the use of that data type with JSON. For example, com.wm.util.Table is not supported for JSON even though it is supported for XML. Integration Server embeds additional type information in XML when converting IData to XML. However, Integration Server cannot embed the additional type information in JSON because the additional type information is treated as JSON text. The resulting JSON text would not match the original JSON text.

Generating Fields from Unquoted Fields in a JSON Object

The JSON standard requires that field names be enclosed in double quotes. However, when creating an document type from a JSON object you may want to create fields for fields in the that are not enclosed in quotes in the source JSON object.

The server configuration parameter watt.server.json.allowUnquotedFieldNames determines whether Integration Server generates fields for unquoted fields. a If watt.server.json.allowUnquotedFieldNames is set `true` and a JSON object with unquoted fields is used as the source for a document type, the resulting document type contains fields that correspond to the unquoted fields as well as the quoted fields. When watt.server.json.allowUnquotedFieldNames is set `false` and a JSON object with unquoted fields is used as the source for a document type, Designer throws an exception and does not create the document type.

Creating an IS Document Type from a JSON Object

Perform the following procedure to create an IS document type from a JSON object.

To create an IS document type from a JSON object

1. In the Service Development perspective, select **File > New > Document Type**.
2. In the New Document Type dialog box, select the folder in which you want to save the IS document type.
3. In the **Element name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
4. Click **Next**.
5. On the Select the Source Type panel, select **JSON** and click **Next**.
6. On the Select a Source Location panel, under **Source location**, select **File/URL**.
7. Enter the path to and name of the JSON object or click **Browse** to navigate to and select the source file.
8. Click **Next**.
9. On the Select Java Wrapper Type panel, under **Java wrapper type for real numbers** select how Integration Server should map real numbers from the JSON object to fields in the IS document type as follows:

Select	To convert real numbers in the JSON object to...
Double	Double Java wrapper type.

Select	To convert real numbers in the JSON object to...
Float	Float Java wrapper type.

Note:
The default setting for **Java wrapper type for real numbers** is set by the watt.server.json.decodeRealAsDouble server configuration parameters. For example, if watt.server.json.decodeRealAsDouble is set to true, Designer displays **Double** as the default for **Java wrapper type for real numbers**. You can override this setting by selecting **Float**.

For more information about the watt.server.json.decodeRealAsDouble server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

10. On the Select Java Wrapper Type panel, under **Java wrapper type for integers** select how Integration Server should map integers from the JSON object to the fields in the IS document type as follows:

Select	To convert integers in the JSON object to...
Long	Long Java wrapper type.
Integer	Integer Java wrapper type.

Note:
The default setting for **Java wrapper type for integers** is set by the watt.server.json.decodeIntegerAsLong server configuration parameter. For example, if watt.server.json.decodeIntegerAsLong is set to true, Designer displays **Long** as the default for **Java wrapper type for integers**. You can override this setting by selecting **Integer**.

For more information about the watt.server.json.decodeIntegerAsLong server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

11. Click **Finish**. Integration Server creates the document type. Designer refreshes the Package Navigator view automatically and displays the new document type.

Creating an IS Document Type from a Broker Document Type

You can create an IS document type from a Broker document type in Designer, if the Integration Server is connected to a Broker. The IS document type retains the structure, fields, data types, and publication properties defined in the Broker document type.

When you create an IS document type from a Broker document type, you create a publishable document type. A *publishable document type* is an IS document type with specific publishing properties, such as storage type and time to live. Additionally, a publishable document type on an Integration Server is bound to a Broker document type. An instance of a publishable document type can be published to a Broker or locally within an Integration Server. For more information about publishable document types, see "Working with Publishable Document Types" on page 553.

To create an IS document type from a Broker document type

1. In the Service Development perspective, select **File > New > Document Type**

2. In the New Document Type dialog box, select the folder in which you want to save the IS document type.

3. In the **Element name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

4. Click **Next**.

5. On the Select a Source Type panel, select **Broker Document Type**, and click **Next**.

   Note:
   The **Broker Document Type** option is enabled only if your Integration Server is connected to a Broker.

6. On the Select a Broker Document Type panel, do one of the following to specify the source file for the document type:

   a. Select the Broker document type from which you want to create an IS document type, from the displayed list of Broker document types on the Broker territory to which the Integration Server is connected.

      You can also type a search string in the **Enter Broker document type name** field to filter the list of Broker document types.

   b. If you want to replace existing elements in the Integration Server namespace with identically named elements referenced by the Broker document type, select the **Overwrite existing elements when importing referenced elements** check box.

      Important:
      Overwriting the existing elements completely replaces the existing element with the content of the referenced element. Any elements on the Integration Server that depend on the replaced element, such as flow services, IS document types, and specifications, might be affected. For more information about overwriting referenced elements, see "Importing and Overwriting References During Synchronization" on page 588.

7. Click **Finish**. Designer refreshes the Package Navigator view automatically and displays the new document type.

Notes:

- When you create an IS document type from a Broker document type that references other elements, Designer will also create an element for each referenced element. Integration Server will contain a document type that corresponds to the Broker document type and one new

element for each element the Broker document type references. Designer also creates the folder in which the referenced element was located. Designer saves the new elements in the package you selected for storing the new publishable document type.

For example, suppose that the Broker document type references a document type named address in the customerInfo folder. Designer would create an IS document type named address and save it in the customerInfo folder. If a field in the Broker document type was constrained by a simple type definition declared in the IS schema purchaseOrder, Designer would create the referenced IS schema purchaseOrder.

- You can associate only one IS document type with a given Broker document type. If you try to create a publishable document type from a Broker document type that is already associated with a publishable document type on your Integration Server, Designer displays an error message.
- If you did not select the **Overwrite existing elements when importing referenced elements** check box and the Broker document type references an element with the same name as an existing Integration Server element, Designer will not create the publishable document type. For more information about overwriting referenced elements, see "Importing and Overwriting References During Synchronization" on page 588
- In the **Publication** category of the Properties panel, the **Provider definition** property displays the name of the Broker document type used to create the publishable document type. Or, if you are not connected to a Broker, this field displays **Not Publishable**. You cannot edit the contents of this field. For more information about the contents of this field, see "About the Associated Provider Definition" on page 558.
- Once a publishable document type has an associated Broker document type, you need to make sure that the document types remain in sync. That is, changes in one document type must be made to the associated document type. You can update one document type with changes in the other by synchronizing them. For information about synchronizing document types, see "About Synchronizing Publishable Document Types" on page 580.

Creating an IS Document Type from an E-form Template

You can create a publishable IS document type from an e-form template stored on the file system or in a content repository.

Keep the following information in mind when creating an IS document type from an e-form template:

- To use an e-form in a repository as the source, a content repository must be set up and the e-form environment must be deployed to the Integration Server on which you want to create the IS document type. For more information about configuring and deploying an e-forms environment, see *Implementing E-form Support for BPM*.
- When creating an IS document type from an e-form template, Integration Server extracts an XML schema definition from the template and uses the XML schema definition to create the IS document type and its supporting IS schema. Typically, when using an XML schema definition as the source file, Designer prompts you for design-time decisions such as whether or not to expand complex types inline, how to handle element references, and the use of

prefixes. However, when the source file is an e-form template, Designer makes the following design-time decisions:

- Complex types are expanded in-line.
- If the XML schema contains only one reference to a particular global element of complex type, Integration Server represents the element reference as a document defined line.
- Integration Server generates a separate IS document type for a named complex type only if the complex type is referenced or is derived from a referenced complex type.
- The IS document type uses the prefixes defined in the XML schema definition.
- Integration Server assigns the IS schema to a unique schema domain whose name is based on the path to and name of the e-form template.

- When you create the IS document type, make sure to select the root node. This will do the following:
  - Ensure that the IS document type matches the e-form template. If you select a node that is not the root node, the resulting IS document type will not match the structure and content of the e-form template. At run-time, processes or services that receive instances of the e-form template will fail due to the mismatch.
  - Ensure that the fields in the IS document type have the same properties as the corresponding elements in the e-form template. If you select a node that is not the root node, the properties for fields in the IS document type might not match the corresponding e-form template. For example, suppose that the e-form template specifies that a field allows null values and is required. If the root node is not selected, the corresponding field in the IS document type might disallow null values (**Allow null** = false) and indicate the field is optional (**Required** = false).
- For an IS document type created from a source, Designer displays the location of the source in the **Source URI** property. Designer also sets the **Linked to source** property to true which prevents any editing of the document type contents. To edit the document type contents, you first need to make the document type editable by breaking the link to the source. For information about allowing editing of elements derived from a source, see "Allowing Editing of Derived Elements" on page 47.

To create an IS document type from an e-form template

1. In the Service Development perspective, select **File > New > Document Type**
2. In the New Document Type dialog box, select the folder in which you want to save the IS document type.
3. In the **Element name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
4. Click **Next**.

5. On the Select the Source Type panel, select one of the following:

Select...	To...
Adobe LiveCycle E-Form Template	Use an Adobe LiveCycle E-Form Template stored on the file system or My webMethods Server as the source for the document type.
Microsoft InfoPath E-Form Template	Use a Microsoft InfoPath E-Form Template stored on the file system or My webMethods Server as the source for the document type.

6. On the Select a Source Location panel, under **Source location**, select one of the following to specify the location of the e-form template:

Select...	To...
File/URL	Use an e-form template on a file system. Enter the path to and name of the e-form template or click **Browse** to navigate to and select the source file. Click **Next**.
My webMethods Server	Use an Adobe LiveCycle E-Form Template or Microsoft InfoPath E-Form Template in the My webMethods Server content repository. Select the content repository that contains the e-form template and click **Next**.

7. On the Select Processing Options panel, under **Content model compliance**, select one of the following to indicate how strictly Integration Server represents content models from the XML Schema definition in the resulting IS document type.

Select...	To...
Strict	Generate the IS document type only if Integration Server can represent the content models defined in the XML Schema definition correctly. Document type generation fails if Integration Server cannot accurately represent the content models in the source XML Schema definition. Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type from any XML schema definition that contains those items.
Lax	When possible, generate an IS document type that correctly represents the content models for the complex types defined in the XML schema definition. If Integration Server cannot correctly represent the content model in the XML Schema definition in the resulting IS document type,

Select...	To...
	Integration Server generates the IS document type using a compliance mode of None. When you select lax compliance, Integration Server will generate the IS document type even if the content models in the XML schema definition cannot be represented correctly.
None	Generate an IS document type that does not necessarily represent or maintain the content models in the source XML Schema definition. When compliance is set to none, Integration Server generates IS document types the same way they were generated in Integration Server releases prior to version 8.2.

8. If you selected strict or lax compliance, next to **Preserve text position**, do one of the following to specify whether document types generated from complex types that allow mixed content will contain multiple **body* fields to preserve the location of text in instance documents.

   - Select the **Preserve text position** check box to indicate that the document type generated for a complex type that allows mixed content preserves the locations for text in instance documents. The resulting document type contains a **body* field after each field and includes a leading **body* field. In instance documents for this document type, Integration Server places text that appears after a field in the **body*.
   - Clear the **Preserve text position** check box to indicate that the document type generated for a complex type that allows mixed content does not preserve the locations for text in instance documents. The resulting document type contains a single **body* field at the top of the document type. In instance documents for this document type, text data around fields is all placed in the same **body* field.

9. If this document type will be used as the input or output signature of a service exposed as a web service and you want to enable streaming of MTOM attachment for elements of type base64Binary, select the **Enable MTOM streaming for elements of type base64Binary** check box.

   For more information about streaming of MTOM attachments, see the *Web Services Developer's Guide*

10. Click **Next**.

11. On the Select Root Node panel, under **Select the root node**, select the root node for the XML schema definition used in the e-form template.

    The standard name for a root node is as follows:

    - Adobe LiveCycle E-Form Template: xdp
    - Microsoft InfoPath E-Form Template: myFields

Keep in mind that the e-form template developer can change the root node.

On the Select a Source Location panel, Designer displays the path to and name of the XML schema definition extracted from the e-form template in the **File/URL** field. Designer creates a set of temporary files containing the XML Schema definition in the workspace. Designer removes the files after creating the IS document type.

12. Under **Element reference handling**, select one of the following to determine how Integration Server handles references to global elements of complex type:

Select...	To...
Only generate document types for elements with multiple references	Instruct Integration Server to create a separate document type for a referenced element only when the XML Schema definition contains multiple references to that element. If an element is referenced multiple times, Integration Server creates a separate document type for the element. Integration Server replaces each element reference with a document reference field. If an element is referenced only once, Integration Server defines the element in line by replacing the element reference with a document field.
Always generate document types for referenced elements	Instruct Integration Server to always create a separate document type for a referenced element even if it is referenced only once. In the document type, Integration Server replaces each element reference with a document reference field

Note: Integration Server always replaces an element reference to an element declaration of simple type with an inline field of type String.

13. Under **Complex type handling**, select one of the following to indicate how Integration Server handles references to named complex type definitions:

Select...	To...
Expand complex types inline	Use a document field defined in line to represent the content of a referenced complex type definition.
Generate document types for complex types	Create a separate IS document type to represent the content for a referenced complex type definition. The resulting IS document type for the root element represents the element of complex type using a document reference field. In turn, this document reference field refers to the IS document type created for the complex type definition.

Select...	To...
	Integration Server generates a separate IS document type for any types derived from the referenced complex types. For more information about derived types, see "Derived Types and IS Document Types" on page 545.

Note: Integration Server always represents an anonymous complex type using a document field defined inline.

14. If you selected **Generate document types for complex types** and you want to register each document type with the complex type definition from which it was created, select the **Register document type with schema type** check box.

Note:
If you want derived type support for document creation and validation, select the **Register document types with schema type** check box. For more information, see "Registering Document Types with Their Schema Types" on page 547.

15. If you want Integration Server to generate IS document types for all complex types in the XML Schema definition regardless of whether the types are referenced by elements or other type definitions, select the **Generate document types for all complex types in XML Schema** check box.

    If you leave this check box cleared, Integration Server generates a separate IS document type for a complex type only if the complex type is referenced or is derived from a referenced complex type.

16. Click **Finish**.

Notes About IS Document Types Created from E-form Templates

Keep the following points in mind when working with IS document types created from e-form templates:

- When an e-form template serves as the source, Designer creates a publishable IS document type. Designer adds the envelope (*_env*) field to the IS document type automatically. This field is a document reference to the pub:publish:envelope document type.
- If Integration Server is connected to a Broker at the time you create an IS document type from an e-form template, the resulting IS document type will be publishable to the Broker and will have an associated Broker document type.
- After the IS document type exists, any modifications to the content or structure of the IS document type will make it out of sync with the e-form template from which it was created. This makes it unusable with the associated e-form. When an instance of the e-form template is received, it will not match the IS document type

- Any changes to the e-form template after using it to create the IS document type results in the template being out of sync with its document type. If the source e-form template changes, delete the IS document type and the associated IS schemas. Then, recreate the IS document type from the latest version of the e-form template.
- When metadata about an IS document type created from an e-form template is published to the CentraSite registry, CentraSite uses the following name for the asset corresponding to the IS document type: *e-formTemplateName:ISDocumentTypeName*.

  This allows multiple e-form templates to have the same name and for each template to be associated with one or more IS document types. Consequently, when searching the metadata repository for a specific e-form template name, consider using the "Contains" clause to search for the e-form template instead of the "Equals" clause.

For more information about using e-forms, refer to *Implementing E-form Support for BPM*.

Creating a Document Type from a File in webMethods Content Service Platform

You can create an IS document type from any file in the webMethods Content Service Platform. The contents of the resulting IS document type consist of fields for the metadata that the Content Service Platform maintains about the file. Integration Server also adds a field named *contentID*. At run time, the *contentID* field contains a unique identifier for the instance of the content type.

To create an IS document type from a file in the webMethods Content Service Platform

1. In the Service Development perspective, select **File > New > Document Type**
2. In the New Document Type dialog box, select the folder in which you want to save the IS document type.
3. In the **Element name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
4. Click **Next**.
5. On the Select a Source Type panel, select **webMethods Content Service Platform** and click **Next**.
6. In the **From Repository** list, select the content repository that contains the content type template from which you want to create a document type.
7. Click **Next**.
8. If you want to filter the contents of selected repository, type search criteria in the text box.

9. Select the content type from which you want to create a document type and click **Next**.

10. In the **Description** field, type a description for the IS document type. This is optional.

    The description will appear in the **Comment** property for the IS document type. If you do not enter a description, the **Comment** property contains a message indicating the source of the IS document type.

11. Click **Next**.

12. On the Select Processing Options panel, under **Content model compliance**, select one of the following to indicate how strictly Integration Server represents content models from the XML Schema definition in the resulting IS document type.

Select...	To...
Strict	Generate the IS document type only if Integration Server can represent the content models defined in the XML Schema definition correctly. Document type generation fails if Integration Server cannot accurately represent the content models in the source XML Schema definition. Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type from any XML schema definition that contains those items.
Lax	When possible, generate an IS document type that correctly represents the content models for the complex types defined in the XML schema definition. If Integration Server cannot correctly represent the content model in the XML Schema definition in the resulting IS document type, Integration Server generates the IS document type using a compliance mode of None. When you select lax compliance, Integration Server will generate the IS document type even if the content models in the XML schema definition cannot be represented correctly.
None	Generate an IS document type that does not necessarily represent or maintain the content models in the source XML Schema definition. When compliance is set to none, Integration Server generates IS document types the same way they were generated in Integration Server releases prior to version 8.2.

13. If you selected strict or lax compliance, next to **Preserve text position**, do one of the following to specify whether document types generated from complex types that allow mixed content will contain multiple **body* fields to preserve the location of text in instance documents.

    - Select the **Preserve text position** check box to indicate that the document type generated for a complex type that allows mixed content preserves the locations for text in instance

documents. The resulting document type contains a **body* field after each field and includes a leading **body* field. In instance documents for this document type, Integration Server places text that appears after a field in the **body*.

- Clear the **Preserve text position** check box to indicate that the document type generated for a complex type that allows mixed content does not preserve the locations for text in instance documents. The resulting document type contains a single **body* field at the top of the document type. In instance documents for this document type, text data around fields is all placed in the same **body* field.

14. If this document type will be used as the input or output signature of a service exposed as a web service and you want to enable streaming of MTOM attachment for elements of type base64Binary, select the **Enable MTOM streaming for elements of type base64Binary** check box.

    For more information about streaming of MTOM attachments, see the *Web Services Developer's Guide*

15. Click **Next**.

16. On the Select Root Node panel, under **Select the root node**, select the root node from which to create the IS document type.Designer displays the metadata fields from the content type. If the content type is associated with an e-form template, Designer displays those elements as well.

17. Under **Element reference handling**, select one of the following to determine how Integration Server handles references to global elements of complex type:

Select...	To...
Only generate document types for elements with multiple references	Instruct Integration Server to create a separate document type for a referenced element only when the XML Schema definition contains multiple references to that element. If an element is referenced multiple times, Integration Server creates a separate document type for the element. Integration Server replaces each element reference with a document reference field. If an element is referenced only once, Integration Server defines the element in line by replacing the element reference with a document field.
Always generate document types for referenced elements	Instruct Integration Server to always create a separate document type for a referenced element even if it is referenced only once. In the document type, Integration Server replaces each element reference with a document reference field

Note: Integration Server always replaces an element reference to an element declaration of simple type with an inline field of type String.

18. Under **Complex type handling**, select one of the following to indicate how Integration Server handles references to named complex type definitions:

Select...	To...
Expand complex types inline	Use a document field defined in line to represent the content of a referenced complex type definition.
Generate document types for complex types	Create a separate IS document type to represent the content for a referenced complex type definition. The resulting IS document type for the root element represents the element of complex type using a document reference field. In turn, this document reference field refers to the IS document type created for the complex type definition. Integration Server generates a separate IS document type for any types derived from the referenced complex types. For more information about derived types, see "Derived Types and IS Document Types" on page 545.

Note: Integration Server always represents an anonymous complex type using a document field defined inline.

19. If you selected **Generate document types for complex types** and you want to register each document type with the complex type definition from which it was created, select the **Register document type with schema type** check box.

Note:
If you want derived type support for document creation and validation, select the **Register document types with schema type** check box. For more information, see "Registering Document Types with Their Schema Types" on page 547.

20. If you want Integration Server to generate IS document types for all complex types in the XML Schema definition regardless of whether the types are referenced by elements or other type definitions, select the **Generate document types for all complex types in XML Schema** check box.

    If you leave this check box cleared, Integration Server generates a separate IS document type for a complex type only if the complex type is referenced or is derived from a referenced complex type.

21. Click **Finish.**

Notes:

- When a content type in the Content Service Platform serves as the source, Designer creates a publishable IS document type. Designer adds the envelope (*_env*) field to the IS document type automatically. This field is a document reference to the pub:publish:envelope document type.
- If Integration Server is connected to a Broker at the time you create an IS document type from an content type in the Content Service Platform, the resulting IS document type will be publishable to the Broker and will have an associated Broker document type.

Creating a Document Type from a Flat File Schema

You can create an IS document type from a flat file schema that resides on the same Integration Server. You can use this IS document type to represent the structure of the flat file. This can be helpful when mapping to or from services that consume or produce flat files.

You can also use the pub.flatFile:generate:createDocumentType to create an IS document type from a flat file schema.

To create a document type from a flat file schema

1. In the Package Navigator of Designer, open the flat file schema from which you want to create an IS document type.
2. On the Flat File Schema editor toolbar, click .

   Integration Server creates an IS document type named *flatFileSchema*DT in the same location as the flat file schema.
3. Select **File > Save**.

Determining How to Represent Complex Types in Document Types

Integration Server processes complex types from an XML Schema in one of two ways, depending on an option you select when you create a new IS document type. One way is to expand the complex type as an "inline" document in the editor. The other way is to generate a separate IS document type for each complex type in the schema, with references to those document types.

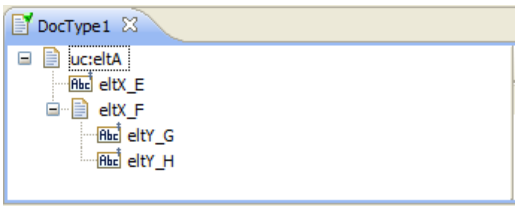
Example XML Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://usecases/xsd2doc/01"
       xmlns:uc="http://usecases/xsd2doc/01" >
   <xsd:element name="eltA" type="uc:documentX" />
   <xsd:complexType name="documentX">
      <xsd:sequence>
         <xsd:element name="eltX_E" type="xsd:string" />
         <xsd:element name="eltX_F" type="uc:documentY" />
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="documentY">
```

```
      <xsd:sequence>
         <xsd:element name="eltY_G" type="xsd:string" />
         <xsd:element name="eltY_H" type="xsd:string" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

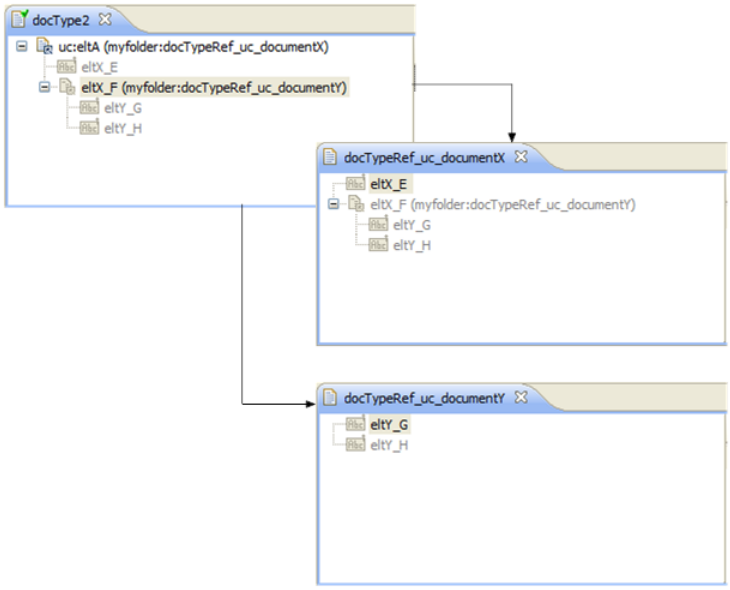
If you select the option to expand complex types inline, the schema processor generates the document type as follows. In this example, the schema processor expanded the complex types named `documentX` and `documentY` inline within the new IS document type:

Complex types expanded inline



If you select the option to generate complex types as separate document types, the schema processor generates the document types as follows. In this example, the schema processor generated three IS document types—one for the complex type named `documentY`, one for the complex type named `documentX` (with a reference to `documentY`), and one for the root element `eltA` (with references to `documentX` and `documentY`):

Complex types generated as separate document types



The schema processor generates all three document types in the same folder.

Note:
If the complex type is anonymous, the schema processor expands it inline rather than generate a separate document type.

If the XML Schema you are using to generate an IS document type contains recursive complex types (that is, element declarations that refer to their parent complex types directly or indirectly), you can avoid errors in the document type generation process by selecting the option to generate complex types as separate document types. (Selecting the option to expand complex types inline will result in infinitely expanding nested documents.)

Derived Types and IS Document Types

In an XML schema definition, new complex types can be *derived* from an existing complex type. The new derived complex types (or simply ***derived types***) are created by either extending or restricting the base complex type. When extended, the derived type contains additional elements or attributes that are not defined for the base type. When restricted, the derived type contains a subset of the original elements or attributes defined for the base type.

When you generate an IS document type from an XML schema definition that contains derived types and you select the **Generate document types for complex types** option, Integration Server creates an IS document type for the base type and one for each derived type. IS document types that represent derived types are referred to as ***derived document types***.

For example, an XML schema might contain a complex type that defines the structure of an address and a derived type that extends the Address complex type to define an address that is specific to the United States:

```
<xsd:element name="purchaseOrder">
     <xsd:sequence>
        <xsd:element name="id"             type="xsd:string"/>
        <xsd:element name="invoiceAddress" type="order:Address">
     </xsd:sequence>
</xsd:element>
<xsd:complexType name="Address">
     <xsd:sequence>
        <xsd:element name="name"   type="xsd:string"/>
        <xsd:element name="street" type="xsd:string"/>
        <xsd:element name="city"   type="xsd:string"/>
     </xsd:sequence>
  </xsd:complexType>
 <xsd:complexType name="USAddress">
     <xsd:complexContent>
        <xsd:extension base="order:Address">
           <xsd:equence>
              <xsd:element name="state"/>
              <xsd:element name="zip"/>
           </xsd:sequence>
        </xsd:extension>
     </xsd:complexContent>
  </xsd:complexType>
```

If you generate an IS document type from this XML schema definition and you select the **Generate document types for complex types** option, Integration Server creates an IS document type for the base Address complex type and another for the derived USAddress complex type.

When data conforms to the derived version rather than the base, an XML document or IData object should indicate the specific derived version that is in use:

- **In an XML document**, the `xsi:type` attribute is included to specify the derived type being used for a complex type. For example, the following XML line indicates that the invoice address will use the alternate format defined by the USAddress complex type:

  ```
  <invoiceAddress xsi:type="order:USAddress">
  ```

- **In a document (IData object)**, Integration Server uses the **doctype* field, which contains the name of the derived document type that represents the structure of a Document field.

*doctype Fields in IS Document Types and Document Fields

When you generate an IS document type from an XML schema definition that contains derived types and you select the **Generate document types for complex types** option, Integration Server creates an IS document type for the base type and one for each derived type. If you also select the **Register document types with schema type** option, Integration Server registers the generated IS document types with the XML schema types. For more information about derived types, see "Derived Types and IS Document Types" on page 545. For more about registration, see "Registering Document Types with Their Schema Types" on page 547.

When IS document types are registered with their XML schema types, Integration Server adds a **doctype* field to the IS document type. Additionally, at run time when the pipeline contains a Document field that conforms to the base IS document type, Integration Server adds the **doctype* field to the Document field. Integration Server uses the **doctype* field when converting IData objects to XML, converting XML to IData objects, and validating IData objects.

*doctype Fields in IS Document Types

When IS document types are registered with their XML schema types, Integration Server adds a **doctype* field to the IS document type of both the base document type and derived document types. When using the IS document type in Designer, Designer lists the valid values you can select for the **doctype* field.

- For a base document type, the list contains the fully-qualified names of all the derived document types that can be substituted for the base type.
- For a derived document type, the list contains only the fully-qualified name of the derived document type itself.

The **doctype* field is of use when using an IS document type for a base type. Integration Server uses the derived document type you select to validate documents (IData objects) against the IS document type. It also uses the derived document type you specify when converting documents (IData objects) to XML. As a result, the derived document type that you select in Designer should correspond to the schema type name that Integration Server should use for the `<xsi:type>` attribute in the XML.

For example, you might have a Document field for an invoice address. To indicate that the structure of the invoice address uses a derived type that represents an address in the United States, for the **doctype* field select the name of the appropriate derived document type (for example, docType_Ref_order_USAddress).

*doctype Fields in Document Fields

When converting XML to a document (IData object), a complex type in the XML is represented in the IData object as a Document field. During the conversion while parsing the XML, Integration Server checks complex types for the `xsi:type` attribute. An `xsi:type` attribute specifies that the type in the instance document overrides the type definition that was defined in the schema. When Integration Server converts a complex type that corresponds to a derived type, it adds the **doctype* field to the Document field in the IData object. Integration Server sets the **doctype* field to indicate that the Document field is an instance of a derived type instead of the base type. Specifically, Integration Server sets the **doctype* field to the fully-qualified name of the IS document type that corresponds to the value of the schema type name specified in the `<xsi:type>` attribute.

For example, a complex type in XML being converted might include the following:

```
<invoiceAddress xsi:type="order:USAddress"
```

When Integration Server generates the Document field for the invoice address, it will add a **doctype* field and set its value to the fully-qualified name of the derived document type that corresponds to the schema type name "order:USAddress" (for example, orders:docType_Ref_order_USAddress).

When working with a Document field that was converted from XML, do not delete or edit the **doctype* field.

Note:
When converting XML to a document (IData object), if Integration Server encounters an `<xsi:type>` attribute in a simple type, Integration Server ignores it. No **doctype* field is added.

Registering Document Types with Their Schema Types

When creating IS document types from XML schema definitions, you can have Integration Server register the IS document types with the XML schema types. To register IS document types, you must select the **Generate document types for complex types** and the **Register document types with schema type** options. By registering the IS document types, you establish a one-to-one mapping between each generated IS document type and its corresponding complex type within the XML schema definition.

Note:
When creating a web service descriptor from a WSDL, Integration Server registers each document type that it creates with the associated schema type defined in the WSDL.

It is important to register IS document types when the XML schema definition uses derived types so that Integration Server can later perform data conversion. That is so that Integration Server can convert data that conforms to the IS document types and the XML schema definition from a document (IData object) to XML, and vice versa. It is also important so that Integration Server can validate documents (IData objects) that use derived types. For more information about derived types and derived document types, see "Derived Types and IS Document Types" on page 545 and "*doctype Fields in IS Document Types and Document Fields" on page 546.

The rest of this section illustrates what happens when Integration Server registers IS document types with their XML schema types and how the registration is used during data conversion. The following shows a portion of an XML schema definition that is used for the illustration.

```
<xsd:element name="purchaseOrder">
     <xsd:sequence>
        <xsd:element name="id"             type="xsd:string"/>
        <xsd:element name="invoiceAddress" type="order:Address">
     </xsd:sequence>
</xsd:element>
<xsd:complexType name="Address">
     <xsd:sequence>
        <xsd:element name="name"   type="xsd:string"/>
        <xsd:element name="street" type="xsd:string"/>
        <xsd:element name="city"   type="xsd:string"/>
     </xsd:sequence>
  </xsd:complexType>
<xsd:complexType name="USAddress">
     <xsd:complexContent>
        <xsd:extension base="order:Address">
           <xsd:sequence>
              <xsd:element name="state"/>
              <xsd:element name="zip"/>
           </xsd:sequence>
        </xsd:extension>
     </xsd:complexContent>
  </xsd:complexType>
```

When you create IS document types from the above XML schema definition, selecting the **Generate document types for complex types** and the **Register document types with schema type** options, Integration Server:

- Creates an IS document type for the base Address complex type and another for the derived USAddress complex type.
- Adds a **doctype* field to IS document types created for the base Address complex type and the derived USAddress complex type.
- Registers the Address complex type with the IS document type it generates for the Address complex type.
- Registers the USAddress complex type with the derived IS document type it generates for the derived USAddress complex type. For example, this might establish a mapping between the complex type order:USAddress and the IS document type docTypeRef_order_USAddress.

Because the IS document types were registered with the XML schema types, Integration Server can later:

- Convert XML data based on the schema to a document (IData object) and validate the document

  When an element in an XML instance is based on a derived type, the XML uses the `xsi:type` attribute to identify the derived type for the element. When the IS document type associated with derived type is registered, Integration Server can locate the correct IS document type to use for the conversion, as well as set the **doctype* field to indicate the IS document type that defines the format in the resulting document (IData object).

For example, if Integration Server converts an XML document that uses the USAddress complex type, when parsing the XML, Integration Server finds the `<invoiceAddress xsi:type="order:USAddress"` element. Integration Server uses the value of the `xsi:type` attribute, that is `order:USAddress`, and looks up the registration to determine the corresponding IS document type. After Integration Server determines the IS document type, it can then do the conversion using the IS document type that corresponds to `order:USAddress`.

During the conversion, Integration Server sets the **doctype* field to the fully-qualified name of the IS document type it found in the registration. As a result, when Integration Server validates the IData object, it determines the correct IS document type to use for validation by using the value in the **doctype* field.

Note:
When converting XML to an IData object, if Integration Server encounters an `<xsi:type>` attribute in a simple type, Integration Server ignores it. No **doctype* field is added.

- Convert Document fields based on the IS document types to XML

  When a Document field is based on a derived document type, the Document field contains a **doctype* field to identify the name of the derived document type that defines the Document field's structure. Integration Server can look up the document type name in the registration to determine the corresponding Qualified Name (QName) of the complex type in the schema. Integration Server uses this QName to populate the `xsi:type` attribute that it places in the XML it is generating.

  For example, if Integration Server converts a Document field containing a **doctype* field that has a value that is the fully-qualified name of the IS document type, "docType_Ref_order_USAddress", it uses the value of the **doctype* field and looks up the registration to determine the corresponding Qualified Name (QName), which it uses to populate the `xsi:type` attribute in the resulting XML.

Generating Fields for Substitution Groups

Integration Server processes substitution group elements in one of two ways, depending on the value of the watt.core.schema.generateSubstitutionGroups property:

- When this property is set to true, the schema processor imports all substitution group members (a non-abstract head element and substitutable elements) as optional fields, even though they are defined as required elements in the XML Schema definition.

  Note:
  Because all the substitution group members are imported as optional, during validation, Integration Server might consider some documents to be valid even though the documents are actually invalid. For example, suppose the original XML schema definition required the head element or one of the member elements to be present. If none of the substitution group elements are present in the instance document, Integration Server considers the document to be valid because the corresponding fields are optional in resulting IS document type. Additionally, if the instance document contains more than one member of the substitution group, Integration Server considers the document to be valid because the corresponding fields are optional.

- When this property is set to false, the resulting document type contains a field that corresponds to the head element in the substitution group, but does not contain any elements for members of the substitution group. This is the default.

When generating fields for a substitution group, Integration Server exhibits the following behavior:

- If the head element is declared as abstract, Integration Server does not include that element in the IS document type.
- Normally, when Integration Server creates a document type for a content model that contains multiple occurrences of an element, Integration Server aggregates the repeated fields into a single array. For example, if Integration Server encounters two elements named "myElement", Integration Servercollects them into a single array named "myElement". However, when Integration Server creates a document type for a substitution group, if the same element is included in the substitution group more than once via two different substitution group members, Integration Server does not aggregate the elements into an array.
- Integration Server cannot create an IS document type from an XML Schema definition that contains a substitution group with a recursive reference to another substitution group. For example, if a member of the substitution group contains a reference to the head element, Integration Server enters a loop which eventually results in a stack overflow error.

*Any Fields in Document Types and Document Fields

If the XML Schema definition used to create an IS document type contains a content model with an `any` element declaration, Integration Server represents the element type contains a content model with a field of type Object named **any*. The `<any>` element declaration in an XML Schema definition acts as placeholder for one or more unknown elements in an instance document. In an `<any>` declaration, the namespace attribute value determines the namespaces to which the matching element can or cannot belong.

At run time, when Integration Server converts an XML node to a document it replaces the **any* field with a field or fields for the replacement elements. Integration Server validates that the replacement element is allowed or not allowed based on the `namespace` attribute value specified for the `<any>` element.

Note:
When Integration Server converts an XML node to a document (IData), Integration Server treats the matching element as if the `processContents` attribute for the `<any>` declaration was set to "`skip`". A value of "`skip`" specifies that the replacement element must be well-formed XML but the Integration Server does not verify that the replacement element is schema-valid.

About Run-time Processing for an IS Document Type that Complies with the Content Model

If the IS document type accurately represents the content model for the complex type from which it was created (the **Model type** property value is not "Unordered"), when Integration Server converts an XML node to a document (IData), Integration Server matches up the contents of an element in the XML node with the content model of the IS document type. If a mismatch occurs

and Integration Server is unable to map the XML node contents to the IS document type, Integration Server appends the remaining data to the resulting document (IData). Integration Server stops attempting to map the XML node content to a field in the IS document type. This mismatch does not result in an error at the time the document is created. However, the document would fail validation by the pub.schema:validate service.

Editing Document Types

When you make a change to an IS document type, keep the following points in mind:

- Any change is automatically propagated to all services, specifications, document fields, and document list fields that use or reference the IS document type. (This happens when you save the updated IS document type to the server.) To view a list of elements that use the IS document type and will be affected by any changes, use the **Find Dependents** command on the right-click menu.
- If you use an IS document type as the blueprint for pipeline or document validation, any changes you make to the IS document type can affect whether the object being validated (pipeline or document) is considered valid.
- The contents of an IS document type with a **Model type** property value other than "Unordered" cannot be modified.
- For an IS document type from a source file such as an XML schema definition or a WSDL document, Designer displays the location of the source file in the **Source URI** property. Designer also sets the **Linked to source** property to true which prevents any editing of the document type contents. To edit the document type contents, you first need to make the document type editable by breaking the link to the source. For information about allowing editing of elements derived from a source, see "Allowing Editing of Derived Elements" on page 47. However, Software AG does not recommend editing the contents of document types created from WSDL documents.

Important Considerations When Modifying Publishable Document Types

Keep the following information in mind when modifying a publishable document type.

- When you modify a publishable document type (for example, delete a field or change a property), the publishable document type may no longer synchronized with the corresponding provider definition (Broker document type or Universal Messaging channel). For information about how to synchronize document types, see "About Synchronizing Publishable Document Types" on page 580.
- Any change is automatically propagated to all services, specifications, document fields, and document list fields that use or reference the IS document type. (This happens when you save the updated IS document type to the server.) To view a list of elements that use the IS document type and will be affected by any changes, use the **Find Dependents** command on the right-click menu.

- If you make the document unpublishable by changing the **Publishable** property to false, publishing services and processes that use the publishable document type will fail. For more information about making a document type unpublishable, see "Making a Document Type Unpublishable" on page 562.
- When you change the messaging connection alias assigned to a publishable document type, you might need to synchronize the publishable document type with its associated provider definition.
- When you change the messaging connection alias assigned to a publishable document type, Integration Server reloads any webMethods Messaging Triggers that subscribe to the publishable document type.
- Changes you make to the contents of a publishable document type might require you to modify the filter for the document type in a trigger condition. For example, if you add, rename, or move fields you need to update any filter that referred to the modified fields. You might also need to modify the service specified in the trigger condition for the webMethods Messaging Trigger.
- When you change the encoding type of a publishable document type, you might need to synchronize the publishable document type with the provider definition on Universal Messaging.

  You might also need to change the provider filters for any webMethods Messaging Triggers that subscribe to the publishable document type. When the encoding type is IData, it is optional to include *_properties* in the provider filter. However, when the encoding type is protocol buffers, you need to include *_properties* in the provider filter. If you want a provider filter that operates on the contents of *_properties* to work regardless of the encoding type, always include *_properties* in the filter expression.
- For an IS document type created from an e-form template, any modifications to the content or structure of the IS document type will make it out of sync with the e-form template from which it was created. This makes it unusable with the associated e-form. When an instance of the e-form template is received, it will not match the IS document type.

About Universal Names and Document Types

Every service and document type on a webMethods Integration Server has a universal name in addition to its regular webMethods name. A *universal name* is a unique public identifier that external protocols (such as SOAP) use to reference a service or document type on a webMethods Integration Server. For more information about assigning a universal name to a document type, see "About Universal Names for Services or Document Types" on page 191.

Printing an IS Document Type

You can use the **View as HTML** command to produce a printable version of an IS document type.

To print an IS document type

1. In Package Navigator view, open the IS document type you want to print.

2. Right-click anywhere inside the Document Type Editor and select **View as HTML**.

   Designer expands any document and document list fields in the IS document type, generates HTML content, and displays the HTML in a new editor.

   If the **Display Properties** check box is selected on the HTML Generation preference page in Service Development preferences, Designer includes the properties for the document type and all of the variables in the document type. You can expand/collapse the properties for individual variables or the parent document type. You can also click **Expand all properties** or **Collapse all properties** to expand or collapse properties for the document type and its contents.

   For more information about HTML generation preferences, see "HTML Generation Preferences" on page 945.

3. To print the HTML page, right-click anywhere in the editor and select **Print**.

   Note: Designer prints the contents of the editor only. Variables and properties that are collapsed will not be expanded in the printed version of the HTML.

Working with Publishable Document Types

A publishable document type is an IS document type with specified publication properties such as storage type, time-to-live, and a message connection alias. In an integration solution that uses the publish-and-subscribe model, services publish instances of publishable document types, and triggers subscribe to publishable document types. A trigger specifies a service that the Integration Server invokes to process the document.

For example, you might create a publishable document type named EmpRec that describes the layout of an employee record. You might create a trigger that specifies that Integration Server should invoke the addEmployeeRecord service when instances of the EmpRec are received. When a service or adapter notification publishes a document of type EmpRec, that document would be queued for the subscribers of document type EmpRec. Integration Server would pass the document to the subscribing trigger and invoke the addEmployeeRecord service.

In a business process, a published document can start or join a process.

In a publication environment that includes Broker or Universal Messaging, each publishable document type is associated with a messaging provider. You create an association between a publishable document type and messaging provider by assigning a messaging connection alias to the document type. A messaging connection alias specifies the configuration information necessary to create a connection to the messaging provider. The messaging connection alias assigned to a publishable document type determines the messaging provider that will receive and route published documents

The messaging connection alias is used by:

- A publishing service to publish an instance of the publishable document type to the messaging provider

- A webMethods Messaging Trigger to retrieve published instances of the publishable document type from the messaging provider.

Once you assign a messaging connection alias to a publishable document type, the messaging provider creates a provider definition, that corresponds to the publishable document type. On Broker, the corresponding provider definition is a Broker document type. On Universal Messaging, the corresponding provider definition is a channel. A publishable document type and its corresponding provider definition must remain in sync. Designer provides tools that you can use to synchronize a publishable document type with its provider definition.

If the publishable document type uses Universal Messaging as the messaging provider, you can specify an encoding type of IData or protocol buffers for instances of the publishable document type. Integration Server uses the encoding type to serialize and deserialize published and received documents. The encoding type you select determines the filtering that the Universal Messaging can preform prior to enqueuing a message for subscribers. For more information, see "About the Encoding Type for a Publishable Document Type" on page 563.

When you build an integration solution that uses publication and subscription, you need to create the publishable document types before you create triggers, services that process documents, and services that publish documents.

Making a Document Type Publishable

Keep the following points in mind when making a document type publishable:

- You can only make an IS document type publishable if you own the lock on the IS document type (or you have it checked out) and you have write permission to the IS document type.
- If you want to publish instances of the document type to a webMethods messaging provider (Broker or Universal Messaging), make sure that a messaging connection alias exists for the provider. If no messaging connection alias exists, all publishable document types will be publishable locally only.
- Instances of a publishable document type that uses Universal Messaging the messaging provider cannot be published locally.
- If an IS document type contains a field named *_env*, you need to delete that field before you can make the IS document type publishable. For more information about the *_env* field, see "About the Envelope Field" on page 560.
- Broker prohibits the use of certain field names, for example, Java keywords, @, *, and names containing white spaces or punctuation. If you make a document type publishable and it contains a field name that is not valid on the Broker, you cannot access and view the field via any Broker tool. Additionally, Broker cannot apply a filter to the contents of the field. However, the Broker transports the contents of the field, which means that any other Integration Server connected to that Broker has access to the field as it was displayed and implemented on the original Integration Server. Use field names that are acceptable to the Broker. See *Administering webMethods Broker* for information on naming conventions for Broker elements.
- The protocol buffers encoding type, which can be used with publishable document types that work with Universal Messaging, does not support certain field names or data types. These

fields cannot be represented in protocol buffers and will be passed through to Universal Messaging . Universal Messaging cannot filter on the contents of these fields. However, subscribing triggers that receive the document will decode the field and include it in the pipeline. For more information about protocol buffers and supported field names and data types. see "Using Protocol Buffers as the Encoding Type" on page 564

- An IS document type that contains a Document Reference field or Document Reference list field named "group" cannot become a publishable document type with an encoding type of protocol buffers. The document type can only use the IData encoding type. The name "group" is a reserved name for protocol buffers.
- If a document type contains a *_properties* field at the top-level and the associated messaging provider is Universal Messaging, Integration Server and Universal Messaging treat the contents of *_properties* as custom header fields in the published document. For more information about the *_properties* field, see "About the Properties Field" on page 560.
- Designer makes an IS document type generated from an e-form template a publishable document type automatically.
- You can make a document type publishable when the **Linked to source** property is set to true. When a document type is linked to its source, you cannot change the structure or contents of the document type. However, Designer does not consider the addition of the *_env* field to be a structural change that breaks the association with the source file.

To make a document type publishable

1. In the Package Navigator view of the Service Development perspective, double-click the document type that you want to make publishable.

   The document type opens in the Document Type Editor window.

2. In the Properties view, set the **Publishable** property to **True**.

3. Next to **Connection alias name**, do one of the following:

   - Select the name of the messaging connection alias with which instances of this document type will be published.
   - Select DEFAULT (*defaultAliasName*) to use the default messaging connection alias.
   - Leave the **Connection alias name** property blank to use the default messaging connection alias.
   - To publishable instances of this document locally only, select IS_LOCAL_CONNECTION.

     Note:
     You can publish a document associated with a Broker connection alias locally by setting the *local* input parameter of the publishing service to true.

4. If you selected a Universal Messaging connection alias for the **Connection alias name** property or you selected DEFAULT and the default messaging connection alias is a Universal Messaging

connection alias, next to **Encoding type**, select one of the following to indicate the format used to encode and decode instances of this publishable document type.

Select...	To encode and decode published documents as...
IData	A serialized IData object. An IData object is the universal container that Integration Server uses to receive input and deliver output. An IData object contains an ordered collection of key/value pairs. When a document is encoded as IData, triggers that subscribe to the document type can specify provider filters for the *_properties* header only.
Protocol buffers	A protocol buffer. Protocol buffers is an approach to encoding and decoding structured data developed by Google. This is the default. When a document is encoded as a protocol buffer, triggers that subscribe to the document type can specify provider filters for the body of the message only. Note that the body of the message includes the headers as well.

For more information about setting the encoding type, see "About the Encoding Type for a Publishable Document Type" on page 563.

5. Next to the **Discard** property, select one of the following to indicate how long instances of this publishable document type remain on the provider before the messaging provider discards them.

Select...	To...
False	Specify that the messaging provider should never discard instances of this publishable document type.
True	Specify that the messaging provider should discard instances of this publishable document type after the specified time elapses. In the fields next to **Time to live** specify the time-to-live value and time units.

6. Next to the **Storage type** property, select the storage method to use for instances of this publishable document type.

Select...	To...
Volatile	Specify that instances of this publishable document type are volatile. Volatile documents are stored in memory.
Guaranteed	Specify that instances of this publishable document type are guaranteed. Guaranteed documents are stored on disk.

For more information about selecting a storage type, see "Setting the Document Storage Type for a Publishable Document Type" on page 571.

7. Select **File > Save**. Designer displays beside the document type name in the Package Navigator to indicate it is a publishable document type.

8. If you selected protocol buffers as the encoding type and a field in the publishable document type cannot be represented in protocol buffers, Designer displays a warning message to that effect. Click **OK** to dismiss the message.

Notes:

- In the **Connection alias type** property, Designer displays **Broker** or **Universal Messaging** to indicate which messaging provider is used by the selected alias.
- In the Properties view, the **Provider definition** property displays the name of the corresponding object created on the messaging provider.
  - Universal Messaging creates a channel that corresponds to the document type. The channel name uses the following naming convention: wm/is/folderName/subFolderName/documentTypeName. If a channel with that name already exists, Integration Server does not create a new channel.

    Note:
    When the Universal Messaging server is in a cluster, Universal Messaging creates the channel on all of the servers in the cluster.
  - Broker has a Broker document type that corresponds to the publishable document type. The Broker document type uses the following naming convention: wm::is::folderName::documentTypeName. If a document type with this name already exists on the Broker, Integration Server appends "_1" to the Broker document type name.
- If the messaging connection alias specified in the **Connection alias name** property is not enabled at the time you make the document type publishable, one of the following occurs:
  - If the **Connection alias type** is **Broker**, the publishable document type can be used only in local publishes. The **Provider definition** property displays "Publishable Locally Only". Later, when the messaging connection alias is enabled, you can create a corresponding Broker document type by pushing the document type to the Broker during synchronization.
  - If the **Connection alias type** is **Universal Messaging**, the **Provider definition** property displays the name of the channel. However, the channel may not exist on Universal Messaging. Later, when the messaging connection alias is enabled, you can create a channel by pushing the document type to Universal Messaging during synchronization.
- When you make a document type publishable, the Integration Server adds an envelope field (*_env*) to the document type automatically. When a document is published, Integration Server and/or the messaging provider populate this field with metadata about the document. For more information about this field, see "About the Envelope Field" on page 560.
- If you selected protocol buffers as the encoding type, Integration Server creates a message descriptor for the publishable document type. For more information about using protocol buffers as the encoding type, see "Using Protocol Buffers as the Encoding Type" on page 564.

- Once a publishable document type corresponds to an associated provider definition, you need to make sure that the document type and provider definition remain in sync. You can update one with changes in the other by synchronizing them. For information about synchronizing document types, see "About Synchronizing Publishable Document Types" on page 580
- If you change messaging connection alias assigned to a publishable document type, you might need to synchronize the publishable document type with its associated provider definition.
- Once a document type is publishable, any changes to the content, structure, or properties can impact the corresponding provider definition, subscribing triggers, or publishing services. For more information about editing a publishable document type, see "Important Considerations When Modifying Publishable Document Types" on page 551.

About the Associated Provider Definition

For a document type, the contents of the **Provider definition** property can indicate the following:

- Whether or not the document type is publishable.
- To which messaging provider the document type is published.
- Whether the publishable document type was created from a Broker document type that was itself created from an IS document type.
- Whether the publishable document type was created from a Broker document type created in an earlier version of a webMethods component.
- Whether instances of the publishable document type can be used in local publishes only. If the publishable document type can be used only in local publishes, there is no corresponding provider definition.

The following table lists and describes the possible contents of the **Provider definition** property.

Provider definition property	Description
wm/is/folderName/documentTypeName	The name of the channel on Universal Messaging that corresponds to the publishable document type.
wm::is::folderName::documentTypeName	The name of the Broker document type that corresponds to the publishable document type. The wm::is prefix indicates that the Broker document type was created from an IS document type. (Either the current document type or an IS document type created and made publishable on another Integration Server.) This prefix does not specify which Integration Server the source IS document type is located on. On the Broker, all document types created from an IS document type are located in the is folder, which is a subfolder of the wm folder. The folderName::documentTypeName portion of the name further

Provider definition property	Description
	identifies where the document type is located on the Broker. **Example:** wm::is::customerSync::Customer::updateCustomer Indicates the Broker document type updateCustomer is located in the following series of folders wm::is::customerSync::Customer.
folderName::documentTypeName	The name of the Broker document type that corresponds to the publishable document type. The absence of the wm::is prefix indicates that the publishable document type was generated from a Broker document type created with an earlier version of a webMethods component. **Example:** Customer::getCustomer Indicates the Broker document type getCustomer is located in the Customer:: folder.
Publishable Locally Only	Indicates that instances of the publishable document type can be used in local publishes only. This publishable document type does not have a corresponding Broker document type. When you made this document type publishable, Integration Server was not connected to the Broker specified in the messaging connection alias. **Note:** If you want instances of this publishable document type to be published to the Broker, you need to create a Broker document type for this publishable document type. When the messaging connection alias is enabled, you can create the provider definition by pushing the publishable document type to the provider during synchronization. For more information about synchronizing, see "About Synchronizing Publishable Document Types" on page 580.
Not Publishable	Indicates that this IS document type is not publishable. For information about making an IS document type

Provider definition property	Description
	publishable, see "Making a Document Type Unpublishable" on page 562.

About the Envelope Field

All publishable document types contain an envelope (*_env*) field. This field is a document reference to the pub:publish:envelope document type. The envelope is much like a header in an e-mail message. The pub:publish:envelope document type defines the content and structure of the envelope that accompanies the published document. The envelope records information such as the sender's address, the time the document was sent, sequence numbers, and other useful information for routing and control.

Because the *_env* field is needed for publication, Designer controls the usage of the *_env* field in the following ways:

- You cannot insert an *_env* field in a document type. Designer automatically inserts the *_env* field as the last field in the document type when you make the document type publishable.
- You cannot copy and paste the *_env* field from one document type to another. You can copy and paste this field to the Input/Output tab or into a specification.
- You cannot move, rename, cut, or delete the *_env* field from a document type. Designer automatically removes the *_env* field when you make a document type unpublishable.
- The *_env* field is always the last field in a publishable document type.

For more information about the *_env* field and the contents of the pub:publish:envelope document type, see the *webMethods Integration Server Built-In Services Reference*.

Note:
If an IS document type contains a field named *_env*, you need to delete that field before you can make the IS document type publishable.

About the Properties Field

You can add a *_properties* document field to a publishable document type to have custom header fields added to documents published to Universal Messaging. When a service publishes an instance of the document type, Integration Server adds the contents of *_properties* as name=value pairs to the header. Specifically, Integration Server adds child fields of *_properties* and their specified values to the header. For example, suppose that a publishable document type contains a *_properties* document with a child field named *myField*. If you set *myField* to 5, when Integration Server publishes the document it adds myField='5' to the document header.

A webMethods Messaging Trigger can create a provider filter to be used with the custom header fields. The filter allows the webMethods Messaging Trigger to indicate which documents it wants to receive based on the header contents. Universal Messaging saves the filter along with the subscription to the document type. When Universal Messaging receives an instance of the

publishable document type, Universal Messaging applies the filter to the custom header fields. Universal Messaging enqueues the document for the trigger only if the filter criteria is met.

Note: Integration Server uses the contents of *_properties* as custom header fields document when the document is published to Universal Messaging only. For all messaging providers, Integration Server includes *_properties* in the body of the published document.

For the contents of the *_properties* field to be added to the message header of a published document, *_properties*:

- Must be a Document or Document reference variable.
- Must be at the top-level of the publishable document type. That is, *_properties* cannot be a child of another document in the document type.
- Can include any number of fields.
- Can contain fields of type String.
- Can contain Object fields with a Java wrapper type of:

  java.lang.Boolean

  java.lang.Byte

  java.lang.Character

  java.lang.Double

  java.lang.Float

  java.lang.Integer

  java.lang.Long

  java.lang.Short

  java.util.Date

- Should not contain fields of type Document, Document List, Document Reference, or Document Reference List. When creating the message header, Integration Server ignores the content of fields of these types in *_properties*. Integration Server includes the entire contents of *_properties* the published document, but Integration Server only uses scalar fields that are direct children of *_properties* in the message header.

For information about creating a filter for use with custom header fields, see "Creating a webMethods Messaging Trigger " on page 682.

About Adapter Notifications and Publishable Document Types

Adapter notifications determine whether an event has occurred on the adapter's resource and then sends the notification data to Integration Server in the form of a published document. For example, if you are using the JDBC Adapter and a change occurs in a database table that an adapter

notification is monitoring, the adapter notification publishes a document containing data from the event and sends it to Integration Server.

There are two types of adapter notifications:

- Polling notifications, which poll the resource for events that occur on the resource.
- Listener notifications, which work with listeners to detect and process events that occur on the adapter resource.

Each adapter notification has an associated publishable document type . When you create an adapter notification in Designer, Integration Server automatically generates a corresponding publishable document type. Designer assigns the publishable document type the same name as the adapter notification, but appends PublishDocument to the name. You can use the adapter notification publishable document type in triggers and flow services just as you would any other publishable document type.

The adapter notification publishable document type is directly tied to its associated adapter notification. Integration Server automatically propagates the changes from the adapter notification to the publishable document type. That is, when working in Package Navigator view, Designer treats an adapter notification and its publishable document type as a single unit. If you perform an action on the adapter notification, Designer performs the same action on the publishable document type. For example, if you rename the adapter notification, Designer automatically renames the publishable document type. If you move, cut, copy, or paste the adapter notification Designer moves, cuts, copies, or pastes the publishable document type.

The **Connection alias name** property for the adapter notification publishable document type will initially have the default messaging connection alias that is configured in Integration Server. This means that the default messaging connection alias will be used to publish and receive instances of the adapter notification publishable document type. Any changes to the **Connection alias name** property of the adapter notification publishable document type will be propagated to its associated adapter notification.

For information about how to create and modify adapter notifications, see the appropriate adapter user's guide.

Making a Document Type Unpublishable

You can change any publishable document type to a regular IS document type by making the publishable document type unpublishable. Keep the following points in mind when changing the publication status of a publishable document type:

- webMethods Messaging Triggers, publishing services, and processes can only use publishable document types. When you make an IS document type unpublishable, webMethods Messaging Triggers, publishing services, and steps in a process that use that IS document type will fail.
- If a publishing service specifies the publishable document type and you make the document type unpublishable, the publishing service will not execute successfully. The next time the service executes, Integration Server throws a service exception stating that the specified document type is not publishable.

- You can only change the publishable status of an IS document type if you own the lock on the IS document type (or you have it checked out) and you have write permission to the IS document type.
- If you make a publishable document type unpublishable and the assigned messaging connection alias is not available, the messaging provider will not remove the provider definition automatically. You will need to remove it manually using the Broker interface in My webMethods or the Universal Messaging Enterprise Manager.

To make a publishable document type unpublishable

1. In the Package Navigator view of the Service Development perspective, double-click the document type that you want to make unpublishable.

   The document type opens in the Document Type Editor window.

2. In the Properties view, set the **Publishable** property to **False**.

3. Select **File > Save**.

   Designer displays the Delete Confirmation dialog box. This dialog box prompts you to indicate whether the associated provider definition should be deleted or retained.

4. If you would like to delete the associated provider definition from the messaging provider, select the **Delete associated provider definition** check box.

5. Click **OK**.

About the Encoding Type for a Publishable Document Type

The encoding type indicates how Integration Server encodes and decodes published instances of the document type. For a publishable document type that uses Universal Messaging as the messaging provider, you can specify an encoding type. You can specify one of the following encoding types:

- IData, the universal container in Integration Server for sending and receiving data. When a document type uses IData as the encoding type, Integration Server encodes published instances of the document type as a serialized IData object.
- Protocol buffers, a format for serializing structured data developed by Google and implemented by Integration Server. When a document type uses protocol buffers as the encoding type, Integration Server encodes the published instances of the document type as protocol buffer.

Note:
When a publishable document type uses Broker as the messaging provider, Integration Server always encodes published documents as a Broker Event. Integration Server encodes locally published documents as IData.

The encoding type for a publishable document type also determines the scope of the message to which Universal Messaging applies a provider filter. In turn, this affects the provider filters that you can build for the webMethods Messaging Triggers that subscribe to the document type.

- When IData is the encoding type, Universal Messaging can filter on the custom header fields added via *_properties* only. The provider filter created by a webMethods Messaging Trigger can include *_properties* header fields only.
- When protocol buffers is the encoding type, Universal Messaging can filter on the body of the document only. However, when creating the published document, Integration Server includes the *_properties* headers in the body of the document as well. The provider filter created by a webMethods Messaging Trigger can include body and *_properties* header fields.

For more information about creating filters for use with Universal Messaging, see "Creating a webMethods Messaging Trigger " on page 682.

Note:
You can only specify an encoding type for a publishable document type in Integration Server and Designer versions 9.7 or later. Additionally, the publishable document type must use Universal Messaging version 9.7 or later as the messaging provider.

Using Protocol Buffers as the Encoding Type

If a publishable document type uses Universal Messaging as the messaging provider, you can use protocol buffers as the encoding type. The primary reason to encode published documents as protocol buffers is to leverage the filtering capabilities of Universal Messaging. When protocol buffers is the encoding type, webMethods Messaging Triggers that subscribe to the publishable document type can create provider filters for the body of the document. Note that the body of the published document contains the *_properties* headers and the *_env* field, as well as the document body fields. When a document type uses IData as the encoding type, webMethods Messaging Triggers that subscribe to the publishable document type can create provider filters for the header of the published document only. By creating provider filters for the body as well as the header of the document, triggers can be more selective about which documents they receive.

When you save a publishable document type for which protocol buffers is the encoding type, Integration Server creates a message descriptor that represents the structure of the document type as a protocol buffer. Integration Server saves the message descriptor along with other metadata in the node.ndf file for the publishable document type. When an instance of the publishable document type is published, Integration Server uses the message descriptor to encode the document as a protocol buffer and then sends the document to Universal Messaging. When a trigger receives the published document, Integration Server uses the message descriptor to decode the document from a protocol buffer.

Note:
The message descriptor is a binary representation of the message structure and is not visible within Designer.

When creating the message descriptor, Integration Server includes only fields that can be represented in the protocol buffers format. Not all field names, data types, and structures that are valid for a publishable document type can be represented in the protocol buffer message descriptor.

When publishing a document, Integration Server places fields that cannot be represented in a protocol buffer message descriptor in an UnknownFieldSet. An UnknownFieldSet is a collection of fields that may be present while encoding or decoding the document but are not present in the message descriptor. Integration Server encodes the UnknownFieldSet as a serialized IData byte array. The UnknownFieldSet, which is included in the published document, is passed through to the subscribers. Universal Messaging cannot use provider filters to filter on the contents of the UnknownFieldSet. However, a webMethods Messaging Trigger that receives the document will be able to decode the UnknownFieldSet and include its contents in the pipeline.

If you encode documents as protocol buffers to make use of provider filters for the document body, you may want to delegate as much filtering to the Universal Messaging as possible. If so, make sure the fields on whose contents you want Universal Messaging to filter can be represented within a protocol buffer message descriptor. Universal Messaging can only filter on fields that can be represented in the protocol buffer message format.

The following list identifies limitations for representing a fields in a protocol buffer descriptor:

- Field names must meet the following criteria to be encoded:
  - First character must be a letter (a-z or A-Z).
  - Subsequent characters must be a letter, number, or underscore symbol (_).

  If the field name does not meet the preceding criteria, Designer displays the following message when you save the publishable document type: Cannot create field "*fieldName*" in publishable document type "*publishableDocumentTypeName*"; this field name is not valid for use with protocol buffer encoding. The Universal Messaging provider will transport the field contents as part or the UnknownFieldSet, which will be visible to Integration Server clients only.

  Note: Integration Server reserves the use of field names that begin with the underscore character for Integration Server usage, for example *_env* and *_properties*.
- Fields at the same level that share the same name, such as fields at the top-level of the document type or sibling fields in a Document variable, cannot be encoded with protocol buffers. Integration Server encodes the identically named fields as part of the IData byte array for the UnknownFieldSet. For information about how Integration Server decodes the contents of fields with the same name, see "Decoding Protocol Buffers" on page 567.

  If the publishable document type contains duplicate variables, Designer displays the following message when you save the publishable document type: Cannot create field "*fieldName*" in publishable document type "*publishableDocumentTypeName*"; field with duplicate names are not permitted with protocol buffer encoding.
- Fields must be defined to be data type supported by protocol buffers encoding.
  - String tables cannot be encoded with protocol buffers and will be defined as byte array within the message descriptor and passed through as a serialized IData object.
  - Objects and Object Lists defined to be an unknown Java wrapper type cannot be encoded with protocol buffers. Instead, unknown Objects and Object Lists will be defined as byte array within the message descriptor and passed through as a serialized IData object.

    Note:

> An Object or Object List field is unknown when the **Java wrapper type** property for the fields is set to **UNKNOWN**. For more information about assigning a Java wrapper type to a field, see "Applying Constraints to a Variable" on page 609.

To generate additional logging information in the Sever log when Integration Server creates the protocol buffer descriptor set the logging level for the server log facility **0154 Protocol Buffer Encoding (Universal Messaging)** to Debug or Trace. Increased logging can help you to locate problems that occur during protocol buffer encoding.

Encoding Documents as Protocol Buffers during Document Publishing

When Integration Server publishes an instance of a publishable document type for which the encoding type is set to protocol buffers, Integration Server encodes the document using the message descriptor created for the publishable document type. The message descriptor is a representation of the publishable document type in the protocol buffer format.

Integration Server only encodes fields with names and data that are represented in the message descriptor in the protocol buffer. Fields whose name or data type cannot be protocol buffer encoded will be passed through in the UnknownFieldSet . Integration Server encodes the UnknownFieldSet field as a serialized IData byte array.

For more information about fields that cannot be represented in protocol buffers, see "Using Protocol Buffers as the Encoding Type" on page 564.

However, at the time Integration Server publishes a document, there might be additional fields that cannot be encoded as protocol buffers. Integration Server adds these fields to UnknownFieldSet.

The following contents of a published document will not be encoded as protocol buffers:

- Undeclared fields. Any fields that are in the published document but are not defined in the publishable document type will be added to the UnknownFieldSet. On the subscribing side, Integration Server decodes these undeclared fields and adds them immediately before the *_env* field.
- Fields with a null value. Even if Integration Server can represent the field in protocol buffers, null values cannot be included in protocol buffers. Fields with null values will be added to the UnknownFieldSet. On the subscribing side, Integration Server decodes these fields as null at their original position as defined in the publishable document type.
- Any list field in which one of the elements is a null value. The entire list is encoded as a single serialized IData and placed in the UnknowFieldSet. On the subscribing side, Integration Server decodes the list field into its original position as defined in the publishable document type.

In addition, document encoding can fail if Integration Server encounters an unexpected data type. For example, if publishable document type defines a field named *myString* to be a String but at run time, the data type of *myString* is not an instance of String, Integration Server cannot encode *myString* because it is not the expected data type. In fact, document encoding fails entirely and publication fails with the following error:

Protocol buffer coder cannot handle data type *dataTypeName* for field *fieldName* in document type: *publishableDocumentType*. Error: *errorMessage*

Decoding Protocol Buffers

When a webMethods Messaging Trigger receives a document encoded as a protocol buffer, Integration Server decodes the document using the publishable document type of which the received document is an instance. During decoding, Integration Server uses the message descriptor created for the publishable document type. The message descriptor is a representation of the publishable document type as a protocol buffer. The IData that results from the decoding will be available in the pipeline when the trigger service executes.

Following are notes about how Integration Server handles certain aspects of decoding:

- Any undeclared fields that were in the published document will follow the last defined field and immediately precede the *_env* field in the decoded IData. Undeclared fields, also called unspecified fields, are those fields that existed in the published document but for which there is not a field in the publishable document type.
- If a published document contains multiple fields that reference a single object, Integration Server decodes the fields as distinct objects instead of as a single object with multiple fields referencing the object.

  For example, suppose that a publishable document type has String fields *abc* and *xyz* and is used in a flow service. At some point in the flow service, in a MAP step, *abc* is mapped to *xyz*. As a result, *xyz* references *abc* . The value of *xyz* is the value of *abc*. Any subsequent change to the value of *abc* in the flow service also changes the value of *xyz*. When a document with multiple references is published, encoded, and then decoded, Integration Server replaces the reference from *xyz* to *abc* with the actual value of *abc*. After decoding, *abc* and *xyz* have the same value initially. But if the value of *abc* later changes, it will not affect the value of *xyz*.
- If a list field, such as String List, contains an empty element in the list but later elements in the list contain values, when Integration Server decodes the list, it condenses the list to fill empty elements. Integration Server condenses the list, first to last, leaving no empty elements in the list.

If a published document contains...

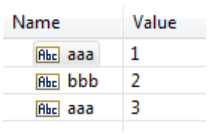
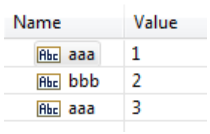
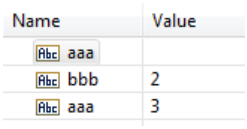
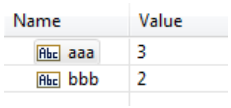
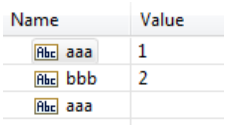
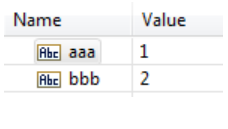
Name	Value
abc	
abc[0]	1
abc[1]	2
abc[2]	
abc[3]	4

Integration Server decodes it as...

Name	Value
abc	
abc[0]	1
abc[1]	2
abc[2]	4

- When decoding values for duplicate fields, Integration Server does not maintain order of the values if one of the fields is empty . Integration Server decodes later occurrence of the field at/in the position of the empty duplicate field.

  The following table provides examples of how Integration Server decodes lists.

If a published document contains...	Integration Server decodes it as...
	
	
	

Name	Value
aaa	1
bbb	2
aaa	3

Name	Value
aaa	1
bbb	2
aaa	3

Name	Value
aaa	
bbb	2
aaa	3

Name	Value
aaa	3
bbb	2

Name	Value
aaa	1
bbb	2
aaa	

Name	Value
aaa	1
bbb	2

In addition, document decoding can fail if Integration Server encounters an unexpected data type. For example, if publishable document type defines a field named *myString* to be a String but, in the received document, the data type of *myString* is not an instance of String, Integration Server cannot decode *myString* because it is not the expected data type. In fact, trigger processing fails with the following error:

Protocol buffer coder cannot handle data type *dataTypeName* for field *fieldName* in document type: *publishableDocumentType*. Error: *errorMessage*

Often, when the above error occurs, it indicates that the publishable document types on the sending and receiving Integration Servers and the provider definition are out of sync. If the publishable document types were the same on the Integration Servers, during encoding, the publishing Integration Server would have caught the mismatch between the data type specified in the publishable document type and the data published in the instance document.

Setting the Encoding Type for a Publishable Document Type

The encoding type determines the format that Integration Server uses to encode and decode instances of this publishable document type. Keep the following points in mind when setting the encoding type:

- You can only set the encoding type for publishable document types that use Universal Messaging 9.7 or later as the messaging provider. Integration Server always encodes publishable document types that use Broker as the messaging provider as a Broker Event. Integration Server encodes locally published documents as IData.
- Integration Server and Designer must be version 9.7 or later.
- The encoding type that you specify affects the scope of filtering that Universal Messaging performs and affects the provider filters you can create for webMethods Messaging Triggers that subscribe to the publishable document type. If you change the encoding type for a publishable document type for which there are already subscribers, the provider filters created in webMethods Messaging Triggers might not work properly.

For more information about encoding type and filters, see " Universal Messaging Provider Filters and Encoding Type" on page 688.

- If you want to use protocol buffers as the encoding type and use a provider filter, the Universal Messaging configuration properties must be set to true, the defaults, on Universal Messaging:
  - **Global Values > ExtendedMessageSelector**
  - **Protobuff Config > Filter ProtobufEvents**

  Use Universal Messaging Enterprise Manager to view and edit the configuration properties for the realm to which Integration Server connects.
- Changing the encoding type causes the publishable document type to be out of sync with the provider definition on Universal Messaging.
- An IS document type that contains a Document Reference field or Document Reference list field named "group" cannot become a publishable document type with an encoding type of protocol buffers. The document type can only use the IData encoding type. The name "group" is a reserved name for protocol buffers.
- You can only set the encoding type for a publishable document type if you own the lock on the IS document type (or you have it checked out) and you have write permission to the IS document type.

To set the encoding type for a publishable document type

1. In the Package Navigator view of the Service Development perspective, double-click the document type for which you want to set the encoding type.

   The document type opens in the Document Type Editor window.

2. In the Properties view, under **webMethods Messaging**, next to **Encoding type**, select one of the following:

Select...	To encode and decode published documents as...
IData	A serialized IData object. An IData object is the universal container that Integration Server uses to receive input and deliver output. An IData object contains an ordered collection of key/value pairs.
Protocol buffers	A protocol buffer. Protocol buffers is an approach to encoding and decoding structured data developed by Google. This is the default.

3. Select **File > Save.**

4. If you selected protocol buffers as the encoding type and a field in the publishable document type cannot be represented in protocol buffers, Designer displays a warning message to that effect. Click **OK** to dismiss the message.

5. Synchronize the publishable document type with its provider definition.

   For information about synchronizing document types, see "About Synchronizing Publishable Document Types" on page 580

About the Type of Document Storage

For a publishable document type, you can set the storage type to determine how Integration Server and the messaging provider store instances of this document. The storage type also determines how quickly the document moves through the webMethods system. You can select one of the following storage types:

- **Volatile storage** specifies that instances of the publishable document type are stored in memory. Volatile documents move through the webMethods system more quickly than guaranteed documents because resources do not return acknowledgements for volatile documents. (An *acknowledgement* indicates that the receiving resource successfully stored or processed the document and instructs the sending resource to remove its copy of the document from storage.) However, if a volatile document is located on a resource that shuts down, the volatile document is not recovered when the resource restarts.

  Integration Server provides at-most-once processing for volatile documents. That is, document delivery and processing are attempted but not guaranteed for volatile documents. Integration Server might process multiple instances of a volatile document, but only if the document was published more than once. Specify volatile storage for documents that have a short life or are not critical.

- **Guaranteed storage** specifies that instances of the publishable document type are stored on disk. Resources return acknowledgements after storing or processing guaranteed documents. Because guaranteed documents are saved to disk and acknowledged, guaranteed documents move through the webMethods system more slowly than volatile documents. However, if a guaranteed document is located on a resource that shuts down, the resource recovers the guaranteed document upon restart.

  webMethods components provide guaranteed document delivery and guaranteed processing (either at-least-once processing or exactly-once processing) for guaranteed documents. *Guaranteed processing* ensures that once a trigger receives the document, it is processed. Use guaranteed storage for documents that you cannot afford to lose.

  Note:
  Some Broker document types have a storage type of Persistent. The Persistent storage type automatically maps to the guaranteed storage type in the Integration Server.

Document Storage Versus Broker Client Queue Storage

The Broker can override the storage type assigned to a document with the storage type assigned to the client queue. A client queue can have a storage type of volatile or guaranteed. Volatile client queues can contain volatile documents only. Guaranteed client queues can contain guaranteed documents and volatile documents.

When the Broker receives a document, it places the document in client queue created for the subscriber (such as a trigger). If the Broker receives a guaranteed document to which a volatile client queue subscribes, the Broker changes the storage type of the document from guaranteed to volatile before placing it in the volatile client queue. The Broker does not change the storage type of a volatile document before placing it in a guaranteed client queue.

The following table indicates how the client queue storage type affects the document storage type.

If document storage type is...	And the client queue storage type is...	The Broker saves the document as...
Volatile	Volatile	Volatile
	Guaranteed	Volatile
Guaranteed	Volatile	Volatile
	Guaranteed	Guaranteed

Note:
On the Broker, each client queue belongs to a client group. The client queue storage type property assigned to the client group determines the storage type for all of the client queues in the client group. You can set the client queue storage type only when you create the client group. By default, the Broker assigns a client queue storage type of guaranteed for the client group created for Integration Servers. For more information about client groups, see *Administering webMethods Broker*.

Setting the Document Storage Type for a Publishable Document Type

Keep the following points in mind when setting the storage type for a publishable document type:

- You can only change the document storage type for a publishable document type if you own the lock on the IS document type (or you have it checked out) and you have write permission to the IS document type.
- For documents published to the Broker, the storage type assigned to a document can be overridden by the storage type assigned to the client queue on the Broker.
- Changing a **webMethods Messaging** property may cause the publishable document type to be out of sync with the associated provider definition. For information about synchronizing document types, see "About Synchronizing Publishable Document Types" on page 580.

To assign the storage type for a publishable document type

1. In Package Navigator view of the Service Development perspective, double-click the publishable document type for which you want to set the storage type.

   The document type opens in the Document Type Editor window.

2. In the Properties view, under **webMethods Messaging**, next to the **Storage type** property, select one of the following:

Select...	To...
Guaranteed	Specify that instances of this publishable document type should be stored on disk.
Volatile	Specify that instances of this publishable document type should be stored in memory.

3. Select **File > Save.**

About the Time-to-Live for a Publishable Document Type

The time-to-live value for a publishable document type determines how long instances of that document type remain on the messaging provider. The time-to-live commences when the Broker receives a document from a publishing Integration Server. If the time-to-live expires before the messaging provider delivers the document and receives an acknowledgement of document receipt, the messaging provider discards the document. This happens for volatile as well as guaranteed documents.

For example, suppose that the time-to-live for a publishable document type is 10 minutes. When the messaging provider receives an instance of that publishable document type, the messaging provider starts timing. If 10 minutes elapse and the messaging provider has not delivered the document or received an acknowledgement of document receipt, the provider discards the document.

For a publishable document type, you can set a time-to-live value or indicate that the messaging provider should never discard instances of the document type.

Setting the Time to Live for a Publishable Document Type

Keep the following points in mind when setting the time-to-live for a publishable document type:

- You can only change the time-to-live for an IS document type if you own the lock on the IS document type (or you have it checked out) and you have write permission to the IS document type.
- Changing a **webMethods Messaging** property causes the publishable document type to be out of sync with the associated provider definition. For information about synchronizing document types, see "About Synchronizing Publishable Document Types" on page 580.

To set a time-to-live value for a publishable document type

1. In Package Navigator view of the Service Development perspective, double-click the publishable document type for which you want to set a time to live.

The document type opens in the Document Type Editor window.

2. In the Properties view, under **webMethods Messaging**, next to the **Discard** property, select one of the following:

Select...	To...
False	Specify that the messaging provider should never discard instances of this publishable document type.
True	Specify that the messaging provider should discard instances of this publishable document type after the specified time elapses. In the **Time to live** property, specify the time-to-live value and units in which the time should be measured.

3. Select **File > Save.**

About Run-Time Validation for a Published Document

In a publish-and-subscribe solution, Integration Server validates a published document against the associated publishable document type. Validation occurs immediately after the publishing service executes. If Integration Server determines that the published document is invalid (that is, the published document does not conform to the associated publishable document type), the publishing service returns a service exception that indicates the validation error. Integration Server does not publish the document.

While document validation ensures that document subscribers receive valid documents only, it can be an expensive operation in terms of resources and performance. In some situations, you might not want to validate the published document. For example, you might want to disable document validation when publishing documents that were already validated. Suppose that a back-end resource, created and validated the document and then sent it to Integration Server. If Integration Server in turn, publishes the document to Broker, you might not need to validate the document when publishing it because it was already validated by the back-end resource. You might also want to disable all document validation when publishing native Broker events.

Integration Server provides two settings that you can use to configure validation for published documents.

- A global setting named watt.server.publish.validateOnIS that indicates whether Integration Server always performs validation, never performs validation, or performs validation on a per document type basis. You can set this property using Integration Server Administrator. For more information about setting this property, see *webMethods Integration Server Administrator's Guide*.
- A webMethods messaging property for publishable document types that indicates whether instances of a publishable document type should be validated. Integration Server honors the value of this property (named **Validate when published**) only if the watt.server.publish.validateOnIS is set to `perDoc` (the default).

Note:
When deciding whether to disable document validation, be sure to weigh the advantages of a possible increase in performance against the risks of publishing, routing, and processing invalid documents.

Specifying Document Validation for Instances of a Publishable Document Type

You can only change the **Validate when published property** for an IS document type if you own the lock on the IS document type (or you have it checked out) and you have write permission to the IS document type.

To specify validation for instances of a publishable document type

1. In Package Navigator view of the Service Development perspective, double-click the publishable document type for which you want to specify validation.

   The document type opens in the Document Type Editor window.

2. In the Properties view, under **webMethods Messaging**, set the **Validate when published** property to one of the following:

Select...	To...
True	Perform validation for published instances of this publishable document type. This is the default.
False	Disable validation for published instances of this publishable document type.

3. Select **File > Save**.

Deleting Publishable Document Types

Before you delete a publishable document type, keep the following in mind:

- When you delete a publishable document type that has a corresponding provider definition, you can choose whether to delete both the publishable document type and the provider definition or just the publishable document type.
- If you intend to delete the associated provider definition as well, make sure that the connection alias to the messaging provider is enabled.
- If the Integration Server is a member of a cluster or the client prefix for the associated messaging connection alias is shared, you can delete the publishable document type but you cannot remove the associated provider definition.

- You can only delete a publishable document type if you own the lock (or have the publishable document type checked out) and have Write permission to it.
- If you delete a Broker document type that is required by another Integration Server, you can synchronize (push) the document type to the Broker from that Integration Server. If you delete a Broker document type that is required by a non-IS Broker client, you can recover the document from the Broker .adl backup file. See *Administering webMethods Broker* for information about importing .adl files.

To delete a publishable document type

1. In Package Navigator view, select the document type you want to delete.
2. Select **Edit > Delete**.

   If you enabled the deleting safeguards in the Preferences dialog box, and the publishable document type is used by other elements, Designer displays a dialog box listing all dependent elements, including triggers and flow services. For information about enabling safeguards to check for dependents when deleting an element, see "Configuring Dependency Checking for Elements" on page 45.

   If the document type is associated with a provider definition, Designer displays the Delete Confirmation dialog box.
3. If you would like to delete the associated provider definition from the messaging provider, select the **Delete associated provider definition** check box.
4. Do one of the following:

Click...	To...
Continue	Delete the element from the Integration Server. References in dependent elements remain.
Cancel	Cancel the operation and preserve the element in the Integration Server.
OK	Delete the element from Integration Server. (The **OK** button only appears if the publishable document type did not have any dependents.)

About Testing Publishable Document Types

To test a publishable document type in Designer, you must create a launch configuration for the document type. In the launch configuration, you can specify input values and publishing method. Designer uses this information to create an instance of the publishable document type. Integration Server then publishes the document locally or to a messaging provider. Designer displays the results of the publish in the Results view.

Testing a publishable document type provides a way for you to publish a document without building a service that does the actual publishing. By testing a publishable document type you

can also test the webMethods messaging triggers that subscribe to the document type. If you select a publication action where you wait for a reply document, you can verify whether or not reply documents are received.

Creating a Launch Configuration for a Publishable Document Type

Use the following procedure to create a re-usable launch configuration that you can run to test a publishable document type.

To create a launch configuration for a publishable document type

1. In the Service Development perspective, select **Run > Run Configurations**.

2. On the Configurations tree, select **IS Document (Publishable)**, and then click .

   A new configuration entry appears below **IS Document (Publishable)** and the launch configuration options appear on the right-hand side of the dialog box.

3. In the **Name** field, enter a new name for your launch configuration. The Publishable tab displays the name of the Integration Server where the document type resides as well as the name of the document type.

4. In the **Integration Server** list, select the Integration Server on which the publishable document type for which you are creating a launch configuration resides.

5. In the **Publishable Document Type** field, click **Browse** to navigate to and select the publishable document type for which you want to create a launch configuration.

6. Click the **Input** tab and specify input values for the publishable document type.

   a. Enter valid values for the fields defined in the publishable document type or click **Load** to retrieve the values from a file. For information about loading input values from a file, see "Loading Input Values" on page 420.

   b. If you want to save the input values that you have entered, click **Save**. Input values that you save can be recalled and reused in later tests. For information about saving input values, see "Saving Input Values" on page 420.

   c. Click **Apply**. When you enter values for constrained objects in the Input tab, Integration Server automatically validates the values. If the value is not of the type specified by the object constraint, Designer displays a message identifying the variable and the expected type.

7. Click the **Action** tab and specify publish settings for the document type.

a. Select the type of publishing for the document.

Note:
The options available here are enabled or disabled, depending on the messaging provider of the publishable document type or the version of Integration Server to which your Designer is connected.

Select...	To...
Publish locally to this Integration Server	Publish an instance of the publishable document type to the same Integration Server to which you are connected.
Publish to the Provider	Publish an instance of this publishable document type to the messaging provider specified by the messaging connection alias assigned to the document type.
Deliver to a destination	Deliver an instance of the publishable document type to a specific destination on the messaging provider. **Note:** This option is disabled if your Designer is connected to a version of Integration Server prior to 9.8 and the messaging provider is Universal Messaging.
Publish locally to this Integration Server and wait for a Reply	Publish an instance of the publishable document type to the same Integration Server to which you are connected and wait for a response document.
Publish to the Provider and wait for a Reply	Publish an instance of this publishable document type to the messaging provider and wait for a response document.
Deliver to a destination and wait for a Reply	Deliver an instance of the publishable document type to a specific destination on the messaging provider and wait for a reply document. **Note:** This option is disabled if your Designer is connected to a version of Integration Server prior to 9.8 and the messaging provider is Universal Messaging.

b. If you selected either **Deliver to a destination** or **Deliver to a destination and wait for a Reply**, in the **Destination ID** field, specify the destination to which you want to deliver the document. You can either enter the destination name or click **Browse** to select the destination. If you click **Browse**, Designer displays all the available Destination IDs.

Note:

If your Designer is connected to Integration Server version 9.8 and the messaging provider is Universal Messaging, the **Browse** button is disabled. You can, however, enter destination ids in the **Destination IDs** field.

Note: Integration Server assigns trigger clients names according to the client prefix set for the Broker connection alias.

c. If you selected a publication action in which you wait for a reply, you need to select the document type that you expect as a reply. You can enter the document type name or click **Browse** to select the document type.

If you click **Browse**, Designer displays all the publishable document types on the Integration Server to which you are currently connected. In the **Elements Name** field, type the fully qualified name of the publishable document type that you expect as a reply or select it from the **Folder** list. If the service does not expect a specific document type as a reply, leave this field blank.

d. Under **Set how long Designer waits for a Reply**, select one of the following:

Select...	To...
Wait indefinitely	Specify that Designer should wait indefinitely for a reply document. Designer will wait for the response for the length of your session on the Integration Server. When you end your session or close Designer, Designer stops waiting for the reply.
Wait for	Specify the length of time that Designer should wait for the reply document. Next to the **Wait for** option, enter how long you want Designer to wait for the reply document.

8. Optionally, click the **Common** tab to define general information about the launch configuration and to save the launch configuration to a file.

9. Click **Apply**.

10. Click **Run** to test the publishable document type now. Otherwise, click **Close**.

Testing a Publishable Document Type

Keep the following points in mind when testing a publishable document type:

- If your publishable document type expects Object variables that do not have constraints assigned or an Object defined as a byte[ ], you will not be able to enter those values in the Input dialog box. To test these values, you must write a Java service that generates input values for

your service and a flow service that publishes the document. Then, create a flow service that first invokes the Java service and then the publishing flow service.

- If you selected a publication action in which you wait for a reply, and Designer receives a reply document, Designer displays the reply document as the value of the *receiveDocumentTypeName* field in the Results view.
- If Designer does not receive the reply document before the time specified for next **Wait for** elapses, Designer displays an error message stating that the publish and wait (or deliver and wait) has timed out. The Results view displays null next to the *receiveDocumentTypeName* field to indicate that the Integration Server did not receive a reply document.

To test a publishable document type

1. In the Package Navigator view of the Service Development perspective, select the publishable document type that you want to test.
2. Select **Run > Run As > Publishable Document**.
3. Do one of the following:
   - If more than one launch configuration exists, select the one you want to run from the Launch Configurations dialog box, and then click **Run**.
   - If a launch configuration does not exist, Designer uses the default launch configuration. When the Input dialog box appears, enter launch configuration data for the test in the Input dialog box and then click **Run**.

     Note:
     The input data is not saved after the default launch configuration runs. If you want to save the input data in a launch configuration, see "Creating a Launch Configuration for a Publishable Document Type" on page 576 for instructions about creating and saving a launch configuration.

Notes:

- Designer displays the instance document and publishing information in the Results view.
- If you selected a publication action in which you wait for a reply, and Designer receives a reply document, Designer displays the reply document as the value of the *receiveDocumentTypeName* field in the Results view.
- If Designer does not receive the reply document before the time specified next **Wait for** elapses, Designer displays an error messages stating that the publish and wait (or deliver and wait) has timed out. The Results view displays null next to the *receiveDocumentTypeName* field to indicate that the Integration Server did not receive a reply document.

About Synchronizing Publishable Document Types

When you synchronize document types, you make sure that a publishable document type matches its associated provider definition. You will need to synchronize document types when:

- You make changes to the publishable document type, including changes to the messaging properties of the publishable document type.
- You made a document type publishable when Integration Server was not connected to the messaging provider (Broker or Universal Messaging).
- You install packages containing publishable document types on Integration Server.
- You make changes to the Broker document type. (This is usually the result of a developer on another Integration Server updating that server's copy of the publishable document type and pushing the change to the Broker document type.)
- You change the client group for the Broker messaging connection alias.

Synchronization Status

Each publishable document type on your Integration Server has a synchronization status to indicate whether it is in sync with the provider definition, out of sync with the provider definition, or not associated with a provider. The following table identifies each possible synchronization status for a document type.

Status	Description
Updated Locally	The publishable document type has been modified on Integration Server.
Updated on the Provider	The publishable document type has been modified on the messaging provider. **Note:** This status applies to publishable document types associated with a Broker connection alias only.
Updated Both Locally and on the Provider	The publishable document type and the provider definition have both been modified since the last synchronization. You must decide which definition is the required one and push to or pull from the Broker accordingly. Information in one or the other document type is overwritten. **Note:** This status applies to publishable document types associated with a Broker connection alias only.
Created Locally	The publishable document type was made publishable when the messaging provider was not connected or the publishable document type was loaded on Integration Server via package replication. An

Status	Description
	associated provider definition may or may not exist on the messaging provider. If the messaging provider is Broker and an associated provider definition exists on the Broker, synchronize the document types by pulling from the Broker. If an associated provider definition does not exist on the messaging provider or the messaging provider is Universal Messaging, create (and synchronize) the provider definition by pushing to the provider.
Removed from Provider	The provider definition associated with the publishable document type was removed from the messaging provider. If you want to recreate the provider definition, push the publishable document type to the provider. If you want to delete the publishable document type on Integration Server and the messaging provider is Broker, pull from the provider. If you want to delete the publishable document type on Integration Server and the messaging provider is Universal Messaging, delete the publishable document type manually. For more information about deleting a publishable document type, see "Deleting Publishable Document Types" on page 574.
In Sync with Provider	The IS document type and the provider definition are already synchronized. No action is required.

Note:
When you switch the Broker configured for the Integration Server to a Broker in a different territory, the Integration Server displays the synchronization status as it was before the switch. This synchronization status may be inaccurate because it does not apply to elements that exist on the second Broker.

Synchronization Actions

When you synchronize document types, you decide for each publishable document type whether to push or pull the document type to the messaging provider. When you push the publishable document type to the messaging provider, you update the provider definition with the publishable document type on your Integration Server. When you pull the document type from the messaging provider, you update the publishable document type on your Integration Server with the provider definition.

The following table describes the actions you can take when synchronizing a publishable document type.

Action	Description
Push to Provider	Update the provider definition with information from the publishable document type.
Pull from Provider	Update the publishable document type with information from the provider definition. **Note:** You can only pull from the messaging provider when the publishable document type uses Broker as the messaging provider.
Skip	Skip the synchronization action for this document type. (This action is only available when you synchronize multiple document types at one time.)

Integration Server does not automatically synchronize document types because you might need to make decisions about which version of the document type is correct. This is especially true when using Broker as the messaging provider. For example, suppose that Integration Server1 and Integration Server2 contain identical publishable document types named Customer:getCustomer. These publishable document types have an associated Broker document type named wm::is::Customer::getCustomer. If a developer updates Customer:getCustomer on Integration Server2 and pushes the change to the Broker, the Broker document type wm::is::Customer::getCustomer is updated. However, the Broker document type is now out of sync with Customer:getCustomer on Integration Server1. The developer using Integration Server1 might not want the changes made to the Customer:getCustomer document type by the developer using Integration Server2. The developer using Integration Server1 can decide whether to update the Customer:getCustomer document type when synchronizing document types with the Broker.

Note:
For a subscribing Integration Server to process an incoming document successfully, the publishable document type on a subscribing Integration Server needs to be in sync with the corresponding document types on the publishing Integration Server and the messaging provider. If the document types and/or provider definition are out of sync, the subscribing Integration Server may not be able to process the incoming documents. In this case, the subscribing Integration Server logs an error message stating that the "Broker Coder cannot decode document; the document does not conform to the document type, *documentTypeName*."

Combining Synchronization Action with Synchronization Status

The effect of a synchronization action on a publishable document type or a provider definition depends on the synchronization status of the publishable document type. The following table describes the result of the push or pull action for each possible document type status.

Note:
You can only pull from the messaging provider when the publishable document type uses Broker as the messaging provider.

Status	Action	Result
Updated Locally	**Push to Provider**	Updates the provider definition with changes made to the publishable document type.
	Pull from Provider	Restores the publishable document type to the previously synchronized version. Any changes made to the publishable document type are overwritten.
Updated on Provider	**Push to Provider**	Restores the provider definition to the previously synchronized version. Any changes made to the provider definition are overwritten.
	Pull from Provider	Updates the publishable document type with changes made to the provider definition (Broker document type).
Updated Both Locally and on the Provider	**Push to Provider**	Updates the provider definition with changes made to the publishable document type. Any changes made to the provider definition prior to synchronization are overwritten.
	Pull from Provider	Updates the publishable document type with changes made to the provider definition (Broker document type). Any changes made to the publishable document type prior to synchronization are overwritten.
Created Locally	**Push to Provider**	If no associated provider definition exists, this action creates an associated provider definition. If an associated provider definition already exists, this action updates the definition with the changes in the publishable document type. **Note:** If publishable document types for this provider definition exist on other Integration Servers, this action changes the synchronization status of those publishable document types to **Updated on the Provider**.
	Pull from Provider	If an associated provider definition exists, this action establishes the association between the document type and the provider definition. If changes have been made to the provider definition, this action updates the publishable document type as well. If an associated provider definition does not exist, this action deletes the publishable document type. **Note:** If publishable document types for this provider definition exist on other Integration Servers, this

Status	Action	Result
		action does not affect the synchronization status of those publishable document types.
Removed from Provider	**Push to Provider**	Recreates the provider definition.
	Pull from Provider	Deletes the publishable document type.
In Sync with the Provider	**Push to Provider**	Pushes the publishable document type to the messaging provider. Even though no changes were made to the provider definition, if other Integration Servers contain publishable document types associated with the provider definition, the status of those publishable document types becomes "Updated on Provider". **Tip:** If a publishable document type is in sync with the provider definition, set the action to **Skip**.
	Pull from Provider	Updates the publishable document type with the provider definition even though no changes are made. **Tip:** If a publishable document type is in sync with the provider definition, set the action to **Skip**.

Note:
For a publishable document type created for an adapter notification, you can select **Skip** or **Push to Provider** only. A publishable document type for an adapter notification can only be modified on the Integration Server on which it was created.

Synchronizing a Single Publishable Document Type

You can synchronize a single publishable document type with its corresponding provider definition. When you synchronize one publishable document type, keep the following points in mind:

- If you want to **Pull from Provider**, you need to have write access to the publishable document type and own the lock on it. For more information about locking elements and access permissions (ACLs), see "About Locking Elements" on page 89 and "Assigning ACLs" on page 82.
- When you **Pull from Provider**, Designer gives you the option of overwriting elements with the same name that already exist on the Integration Server. The provider definition (Broker document type) might reference elements such as an IS schema or other IS document types. If the Integration Server you are importing to already contains any elements with the referenced names, you need to know if there is any difference between the existing elements and those

being imported from the messaging provider (Broker). If there are differences, you need to understand what they are and how importing them will affect any integration solution that uses them. For more information about overwriting existing elements, see "Importing and Overwriting References During Synchronization" on page 588.

- For a publishable document type created for an adapter notification, you can select **Push to Provider** or **Skip** only.
- If the **Linked to source** property is set to true for the publishable document type, the action you can take depends on the source for the publishable document type. You can select:
  - **Pull from Provider** only if the **Source URI** is a Broker document type.
  - **Push to Provider** only if the **Source URI** is a URI other than a Broker document type.

  When **Linked to source** is true, the content and structure of the document type cannot be modified. For information about allowing editing of an element created from a source file, see "Allowing Editing of Derived Elements" on page 47.

To synchronize a single publishable document type

1. In Package Navigator view, select the publishable document type that you want to synchronize.
2. Select **File > Sync Document Types > Selected**. Designer displays the Synchronize dialog box. The Synchronize dialog box displays the synchronization status of the document type, as described in "Synchronization Status" on page 580.
3. Under **Action**, do one of the following:

Select...	To...
Push to Provider	Update the provider definition with the publishable document type.
Pull from Provider	Update the publishable document type with the provider definition. This option is available when the provider definition is a Broker document type only.

Note:
The result of a synchronization action depends on the document status. For more information about how the result of a synchronization status depends on the synchronization status, see "Combining Synchronization Action with Synchronization Status" on page 582.

4. If you select **Pull from Provider**, as the action, Designer enables the **Overwrite existing elements when importing referenced elements** check box.
5. If you want to replace existing elements in the Integration Server with identically named elements referenced by the provider definition (Broker document type), select the **Overwrite existing elements when importing referenced elements** check box. See "Importing and

Overwriting References During Synchronization" on page 588 for more information about this topic.

6. Click **Synchronize** to synchronize the document type and provider definition.

Synchronizing Multiple Document Types Simultaneously

You can synchronize multiple publishable document types with their corresponding provider definitions at one time. For each publishable document type, you can specify the direction of the synchronization. You can push the publishable document type to the messaging provider or pull the provider definition from the messaging provider. If you do not want to synchronize some publishable document types that are out of sync, you can skip them during synchronization.

When synchronizing multiple document types at once, you can do one of the following:

- **Synchronize out-of-sync document types only.** Use this option to view and synchronize all publishable document types that are out sync with their associated provider definition.
- **Synchronize all publishable document types.** Use this option to view and synchronize all publishable document types regardless of sync status.

Keep the following points in mind when synchronizing multiple document types at one time:

- If you want to **Pull from Provider**, you must have write access to the publishable document type. The publishable document type must be either unlocked, or you must have locked it yourself.
- The **Pull from Provider** action is available for publishable document types that use Broker as the messaging provider only. **Pull from provider** can be selected only when the corresponding provider definition is a Broker document type.
- When you pull document types from Broker, Designer gives you the option of overwriting elements with the same name that already exist on the Integration Server. The provider definition (Broker document type) might reference elements such as an IS schema or other IS document types. If the Integration Server to which you are importing already contains any elements with the referenced names, you need to know if there is any difference between the existing elements and those being imported from the Broker. If there are differences, you need to understand what they are and how importing them will affect any integration solution that uses them. For more information about overwriting existing elements, see "Importing and Overwriting References During Synchronization" on page 588.
- For a publishable document type created for an adapter notification, you can only select **Push to Provider** or **Skip**. A publishable document type for an adapter notification can only be modified on the Integration Server on which it was created.
- If the **Linked to source** property is set to true for the publishable document type, the action you can take depends on the source for the publishable document type. You can select:
  - **Pull from Provider** only if the **Source URI** is a Broker document type.
  - **Push to Provider** only if the **Source URI** is a URI other than a Broker document type.

Note:
When synchronizing multiple document types, Designer does not prevent Integration Server from overwriting publishable document types for which **Linked to source** is true.

- When you switch the Broker configured for Integration Server to a Broker in a different territory, Integration Server displays the synchronization status as it was before the switch. This synchronization status may be inaccurate because it does not apply to elements that exist on the second Broker.
- The result of a synchronization action depends on the document status. For more information about how the result of a synchronization status depends on the synchronization status, see "Combining Synchronization Action with Synchronization Status" on page 582.

To synchronize multiple document types

1. In Designer, do one of the following:
   - To view and synchronize only out-of-sync document types, select **File > Sync Document Types > All Out-of-Sync**. Designer displays the Sync All Out of Sync Document Types dialog box.
   - To view and synchronize all document types, regardless of sync status, select **File > Sync Document Types > All**. Designer displays the Sync All Document Types dialog box.
2. If you want to specify the same synchronization action for all of the publishable document types, do one of the following:

Select...	To...
Set All to Push	Change the **Action** for all publishable document types in the list to **Push to Provider**. **Note:** When you select **Set All to Push**, Designer sets the publication action for adapter notification document types to **Skip**.
Set All to Pull	Change the **Action** for all publishable document types in the list to **Pull from Provider**.
Set All to Skip	Change the **Action** for all publishable document types in the list to **Skip**.

3. If you want to specify a different synchronization action for each publishable document type, use the **Action** column to select the synchronization action.

Select...	To...
Push to Provider	Update the provider definition with the publishable document type.
Pull from Provider	Update the publishable document type with the provider definition.

Select...	To...
Skip	Skip the synchronization action for this document type.

4. If you want to replace existing elements in Package Navigator view with identically named elements referenced by the Broker document type, select the **Overwrite existing elements when importing referenced elements** check box. For more information about importing referenced elements during synchronization, see "Importing and Overwriting References During Synchronization" on page 588.

5. Click **Synchronize** to perform the specified synchronization actions for all the listed publishable document types.

Synchronizing Document Types in a Cluster

Universal Messaging and Broker can be used with an Integration Server cluster and a non-clustered group of Integration Servers that receive message from Broker in a load-balanced fashion. A change in a publishable document type on one Integration Server does not automatically result in a change to all Integration Servers in the cluster. You must synchronize each Integration Server with the messaging provider individually.

Synchronizing Document Types Across a Gateway

webMethods does not support synchronization of document types across a gateway. (A gateway connects two Broker territories.) If you set up two or more Broker territories connected by gateways, the only way to synchronize document types is to replicate packages between Integration Servers in each territory. For information about replicating and loading packages from one Integration Server to another see *webMethods Integration Server Administrator's Guide*.

Importing and Overwriting References During Synchronization

When you create a publishable document type from a Broker document type or synchronize a publishable document type by pulling a Broker document type from the Broker, you must decide if you want to overwrite any existing elements associated with the Broker document type.

For example, suppose that you are creating a publishable document type from a Broker document type that was created on another Integration Server. The Broker document type might reference elements such as an IS schema or other IS document types. However, the Integration Server on which you are creating the publishable document type might already contain elements with the referenced names. Before you overwrite the existing elements, you need to know if there are any differences between the existing elements and those being imported from the Broker. If there are differences, you need to understand what they are and how importing them will affect any elements that use them, such as services, IS document types, or triggers.

When you create a new document type from a Broker document type or when you synchronize document types, you can use the **Overwrite existing elements when importing referenced**

elements check box to indicate whether existing elements should be overwritten by imported elements of the same name.

What Happens When You Overwrite Elements on the Integration Server?

If you choose to overwrite existing elements when you are creating a document type or synchronizing, Integration Server does the following when it encounters existing elements with the same names as referenced elements:

- If the Write ACL of a referenced element is set to WmPrivate, the Integration Server skips that element. The Integration Server considers the element to be in sync.
- If the lock can be obtained for all referenced elements and the current user has write permission for the elements, the Integration Server overwrites the existing elements and synchronization (or document type creation) succeeds.

During synchronization, if Integration Server cannot overwrite one of the elements referenced by the Broker document type, the synchronization fails. The Integration Server does not update any of the referenced elements or the publishable document type. Similarly, when you create a publishable document type from a Broker document type, if the Integration Server cannot overwrite one of the elements referenced by the Broker document type, the Integration Server does not create the publishable document type.

What Happens If You Do Not Overwrite Elements on the Integration Server?

If you choose not to overwrite elements when you create a publishable document type from a Broker document type, Integration Server will not create the publishable document type if the Broker document type references elements with the same name as existing element son the Integration Server.

If you choose not to overwrite elements when you synchronize document types by pulling from the Broker, Integration Server does not synchronize any document type that references existing elements on the Integration Server. Integration Server synchronizes only those document types that do not reference elements.

Publishing Documents as JMS Messages

You can create a launch configuration that publishes an instance of an IS document type to a JMS provider. By using a launch configuration to publish the JMS message, you can test the document type in a JMS solution without needing to build a service that sends the document type as JMS message. You can also test any JMS triggers that receive messages from the destination specified in the launch configuration and subsequently any services or business processes that might be invoked by the JMS trigger.

In the launch configuration, you specify the JMS connection alias to use to send the JMS message, the destination for the JMS message, and contents for the document, message header, and message properties. When you run the launch configuration, Integration Server uses the supplied information to create a JMS message in which the document contents become the body of the JMS message. Integration Server sends the JMS message to the JMS provider in the assigned JMS connection alias.

Creating a Launch Configuration to Publish a Document as a JMS Message

Keep the following information in mind when creating a re-usable launch configuration that publishes an instance of an IS document type as a JMS message:

- You can publish any IS document type as a JMS message including publishable document types.
- Before you create the launch configuration, a JMS connection alias that you want the launch configuration to use to send the message must already exist.

Use the following procedure to create a re-usable launch configuration that you can run to publish a document as a JMS message to a JMS provider.

To create a launch configuration to publish a document as a JMS message

1. In Package Navigator view in the Service Development perspective, open the IS document type that you want to publish as a JMS message.

2. Select **Run > Run Configurations** to open the Run Configurations dialog box.

3. On the Configurations tree, select **IS Document (Publish as JMS Message)**, and then click .

   A new configuration entry appears below **IS Document (Publish as JMS Message)** and the launch configuration options appear on the right-hand side of the dialog box.

4. In the **Name** field, enter a new name for your launch configuration. The Document tab displays the name of the Integration Server where the document resides as well as the name of the document that you want to publish.

   You can change the document specified in the **Document Type** field by clicking **Browse**.

5. Click the **JMS Settings** tab and specify the JMS message details:

   a. In the **JMS connection alias name** field, click . In the Select a JMS connection alias for *documentName* dialog box, select the JMS connection alias that you want this launch configuration to use to receive messages from the JMS provider. Click **OK**.

      If a JMS connection alias has not yet been configured on Integration Server, Designer displays a message stating the JMS subsystem has not been configured. For information about creating a JMS connection alias, see *webMethods Integration Server Administrator's Guide*.

   b. In the **Destination name** field, do one of the following to specify the destination:

- If the JMS connection alias uses JNDI to retrieve administered objects, specify the lookup name of the Destination object.
- If the JMS connection alias uses the native webMethods API to connect directly to Broker, specify the provider-specific name of the destination.
- If the JMS connection alias creates a connection on Broker or Universal Messaging, click ... to select from a list of existing destinations. After you select the destination, click **OK**.

c. From the **Destination type** list, do the following:

- Select **Queue** to send the message to a particular queue.
- Select **Topic** to send the message to a topic.

You need to specify a destination type only if you specified a **JMS connection alias name** that uses the native webMethods API.

Note: Designer populates **Destination type** automatically if you selected a destination from the list of existing destinations on the JMS provider.

d. Select **Prepare message for BPM** to add information to the JMS message that enables Process Engine to start a process instance when it receives the message. When this check box is selected, the published JMS message includes the `documentType` property which specifies the fully qualified name of the IS document type used to create the JMS message. Process Engine uses the document type name to map the JMS message to the correct process model and start a process instance.

e. Under **JMS Message Header and Properties**, specify the values for the pre-defined and custom properties that you want to add to the JMS message header. Click to add a row to specify custom properties. Click to insert a row and to delete a row.

6. Click the **Input** tab and specify input values for the document, which will form the contents of the JMS message body.

a. Enter valid values for the fields defined in the document or click **Load** to retrieve the values from a file. For information about loading input values from a file, see "Loading Input Values" on page 420.

b. If you want to save the input values that you have entered, click **Save**. Input values that you save can be recalled and reused in later runs. For information about saving input values, see "Saving Input Values" on page 420.

c. Click **Apply**. When you enter values for constrained objects in the Input tab, Integration Server automatically validates the values. If the value is not of the type specified by the object constraint, Designer displays a message identifying the variable and the expected type.

7. Optionally, click the **Common** tab to define general information about the launch configuration and to save the launch configuration to a file.

8. Click **Apply**.

9. Click **Run** to run the launch configuration to publish the IS document as a JMS message now. Otherwise, click **Close**.

Publishing a Document as a JMS Message

Keep the following information in mind when publishing a document as a JMS message.

- You can publish any IS document type as a JMS message including publishable document types.
- The JMS connection alias that you want to use to send the JMS message must already exist.

To publish a document as a JMS message

1. In Package Navigator view, select the document that you want to publish as a JMS message.

2. In Designer: **Run > Run As > Publish as JMS Message**.

3. Do one of the following:

   - In the Input dialog box, view and edit the input data for the launch configuration, then click **Run**.
   - If more than one launch configuration exists, select the one you want to run from the Select Launch Configuration dialog box, and then click **OK**.
   - If a launch configuration does not exist, Designer creates a new launch configuration. Enter launch configuration data in the Input dialog box and then click **OK**. For more information about the data needed by the launch configuration, see "Creating a Launch Configuration to Publish a Document as a JMS Message" on page 590.

     Note:
     The input data is saved automatically after the launch configuration runs. To save the input data in a launch configuration, see "Saving Input Values" on page 420.

   Designer creates and publishes a JMS message. Designer displays the JMS message in the Results view.

28 Working with XML Document Types

- What Is an XML Document Type? 594
- Why Use XML Document Types Instead of IS Document Types? 594
- Creating an XML Document Type 597

An XML document type is an asset in the IS namespace created from an XML Schema definition. When you create an XML document type from an XML schema definition, Integration Server creates a collection of XML document types to represent the structure, content, and constraints defined in an XML schema definition. Each XML document type corresponds to a global element declaration, global attribute declaration, or global complex type definition in an XML Schema definition.

What Is an XML Document Type?

When you create an XML document type from an XML schema definition, Integration Server creates a collection of assets to represent the XML schema definition, which can include:

- XML document types, each of which corresponds to a global element declaration, global attribute declaration, or global complex type definition in an XML schema definition.
- XML fields each of which corresponds to global element declaration with simple content.
- IS schemas which contain the global simple type definitions for a particular namespace in the XML schema definition.

Like IS document types, XML document types can be used to define the structure of document being created from an XML node or a document being converted to XML, define service signatures, build a document or document list field, or perform data validation. However, XML document types more accurately represent XML Schema definitions and provide support for XML schema constructs/feature that are not supported through IS document types.

What Is XMLData?

At run time, instances of XML document types and XML fields are of contained in an XMLData object. XMLData is an IData object that uses a specific encoding format to represent the XML Information Set (XML Infoset). The format facilitates all the features of XML Infoset and XML Schema, including support for capabilities such as nested model groups and substitution groups. The format also eliminates the need to specify an association between prefixes and namespace URIs.

Unlike the traditional encoding used for XML representation with raw IData, the encoding format for XMLData is not public and is subject to change at any time. The various built-in services that support XMLData, including those in the pub.xmldata folder in the WmPublic package, and flow MAP step operations are the only supported means of accessing and modifying XMLData. Directly manipulating an XMLData object as one can a traditional IData object will lead to unexpected results.

Why Use XML Document Types Instead of IS Document Types?

While XML document types and IS document types have similar uses and, in some cases, similar sources as both can be created from an XML schema definition, XML document types offer the following distinct benefits:

- Improved XML namespace handling. XML document types do not use prefixes from the XML schema definition in the names of document types or fields (variables). Instead IS uses the

following format for XML document type names, XML field names, and names of fields within XML document types: NCName#NamespaceURI. This naming convention ensures that XML document types, XML fields, and fields within a document type have a unique name, preventing the conflicts that arise when IS document types are generated using one set of prefixes and the instance XML documents use a different set of prefixes.

- Nested and repeating model group support. IS can create XML document types from XML schema definitions that contain nested model groups or repeating model groups. An IS document type cannot correctly represent nested and repeating model groups.
- Any attribute support. IS can create XML document types from XML schema definitions that contain the `anyAttribute` element. An IS document type cannot correctly represent the `anyAttribute` element.
- Improved support for substitution groups.
- Improved support for `any` element.
- Support for xsi:nil and xsi:type on simple types (String fields). IS document types support xsi:nil and xsi:type on complex types (Document fields) only.
- Improved handling of identically named fields at the same level.

If you want your solutions to incorporate or leverage any above the above items, consider using XML document types instead of IS document types in your solutions.

Note:
XML document types and instance documents based on XML document types are intended to implement XML Schema and XML as closely as possible. Behavior that is inconsistent with XML Schema and XML will be treated as known issues that need resolution. Implementations should not exploit behavior that is inconsistent with XML and XML schema as it may have unpredictable results.

Differences Between XML Document Types and IS Document Types

In addition to improved namespace handling and support for XML schema constructs such as nested model groups, repeating model groups, and any attribute, XML document types and IS document types have the following differences:

- XML document types can be created from XML schema definitions only.
- Integration Server uses the names in an XML Schema definition to name the generated IS assets, including the XML document types. This is unlike IS document types where you can specify the name of the first IS document type generated from an XML schema definition but Integration Server creates the names of any subsequent IS document types.
- XML document types do not use prefixes from the XML schema definition in the names of XML document types. XML fields, or fields withing XML document types. Instead IS uses the following format for XML document type names, XML field names, and names of fields within

XML document types: *NCName#NamespaceURI* . Integration Server document types use prefixes from the XML schema definition in field (variable) names.

- In XML document types, attribute names do not include the @ symbol to indicate that it is an attribute.
- XML document types do not contain **body*, **doctype*, or **any* fields.
- XML document types provide improved handling of identically named fields at the same level. In XML document types, Integration Server maintains a particle ID for each field. To view the particle ID, hover the cursor over the name of the field. Designer displays properties for the field including the following for the name: {*ID*}*NCName#NamespaceURI* where *ID* is a number representing the occurrence of the field in the document type or the pipeline. For example, {2}myLocalName#myNamespaceName indicates the second occurrence of a field named myLocalName#myNamespaceName.

  In IS document types, all occurrences of identically named fields at the same level are collected into a single array. This approach may not preserve order during runtime.

  Note: Integration Server creates arrays for XML document types when an individual element has a maxOccurs greater than 1. If there are two fields named myLocalName#myNamespaceName and each has a maxOccurs greater than 1, Integration Server creates {1}myLocalName#myNamespaceName as an array and {2}myLocalName#myNamespaceName as an array.
- The contents of XML document types and XML fields cannot be edited.
- A Document Reference or Document Reference List variable contained in an IS document type or in a service signature can reference an XML document type that corresponds to a complex type definition or a root XML document type only.

Limitations of XML Document Type Usage

Although XML document types and IS document types can be used in nearly identical ways, there are some limitations in the usage of XML document type:

- XML document types cannot be made publishable. That is, an XML document type cannot become a publishable document type.
- XML document type cannot be used in web services, which includes the signatures of services used as operations, headers, faults, and the pub.soap.handler* services.
- XML document types cannot be used as the top-level element in a service signature. That is, on the Input/Output tab, you cannot specify an XML document type for the **Input** field or **Output** field.
- XML document types should not be created from an XML schema definition in an Event Type Store.
- XML document types should not be created from e-forms.

Creating an XML Document Type

When you create an XML document type you specify the following:

- The destination folder in which you want Designer to place the generated XML document types, XML fields, IS schemas, and folders.
- The source XML schema definition.
- Whether or not Integration Server use the Xerces Java parser to validate the XML Schema definition before creating XML document types.

There are no additional options, making the process of creating an XML document type less complex than that of creating an IS document type.

When you create an XML document type, keep the following information in mind:

- You can create only one set of XML document types per folder. If you used folderA as the destination for the XML document types and other assets created for mySchema.xsd, you cannot use folderA as the destination for the XML document types and other assets generated from another XML schema definition. However, you could use a subfolder in folderA as the destination for the XML document type and other assets created for another XML schema definition.
- Do not use a folder created by Integration Server to store assets generated for an XML schema definition as the destination folder for new XML document types.
- To create an XML document type from an XML Schema definition in CentraSite, Designer must be configured to connect to CentraSite.

To create an XML document type

1. In the Service Development perspective of Designer, click **File > New > XML Document Type**.
2. In the Create a New XML Document Type dialog box, select the folder in which you want to save the XML document types, XML fields, IS schemas, and folders generated from the XML schema definition.
3. Click **Next**.
4. On the Select a Source Location panel, under **Source location**, do one of the following to specify the source XML schema definition for the XML document type:
   - To use an XML schema definition in CentraSite as the source, select **CentraSite**.
   - To use an XML schema definition that resides on the Internet as the source, select **File/URL**. Then, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
   - To use an XML Schema definition that resides on your local file system as the source, select **File/URL**. Then, type in the path and file name, or click the **Browse** button to navigate to and select the file.

5. If you want Integration Server to use the Xerces Java parser to validate the XML Schema definition, select the **Validate schema using Xerces** check box.

   Note: Integration Server automatically uses an internal schema parser to validate the XML Schema definition. However, the Xerces Java parser provides stricter validation than the Integration Server internal schema parser. As a result, some schemas that the internal schema parser considers to be valid might be considered invalid by the Xerces Java parser.

6. If you selected CentraSite as the source, click **Next**. Then, under Select a Schema, select the XML schema definition that you want to use as the source and click **Finish**.

   If Designer is not configured to connect to CentraSite, Designer displays the **CentraSite**> **Connections** preference page and prompts you to configure a connection to CentraSite.

7. Click **Finish**.

Notes:

- Integration Server uses the internal schema parser to validate an XML schema definition. If you selected the **Validate schema using Xerces** check box, Integration Server also uses the Xerces Java parser to validate the XML Schema definition. With either parser, if the XML Schema does not conform syntactically to the schema for XML Schemas defined in *XML Schema Part 1: Structures* (which is located at http://www.w3.org/TR/xmlschema-1), Integration Server does not create an XML document type. Instead, Designer displays an error message that lists the number, title, location, and description of the validation errors within the XML Schema definition. If only warnings occur, Designer generates the XML document type and the other assets.

  Note: Integration Server uses Xerces Java parser version J-2.11.0. Limitations for this version are listed at http://xerces.apache.org/xerces2-j/xml-schema.html.

- When validating XML schema definitions, Integration Server uses the Perl5 regular expression compiler instead of the XML regular expression syntax defined by the World Wide Web Consortium for the XML Schema standard. As a result, in XML schema definitions consumed by Integration Server, the pattern constraining facet must use valid Perl regular expression syntax. If the supplied pattern does not use proper Perl regular expression syntax, Integration Server considers the pattern to be invalid.

  Note:
  If the watt.core.datatype.usejavaregex configuration parameter is set to true, Integration Server uses the Java regular expression compiler instead of the Perl5 regular expression compiler. When the parameter is true, the pattern constraining facet in XML schema definitions must use valid syntax as defined by the Java regular expression.

29 Working with Specifications

■ Creating a Specification 600

A specification is a "free-standing" element that defines a set of service inputs and outputs. The specification defines the entire service signature. If you have multiple services with the same input and output requirements, you can point each service to a single specification rather than manually specify individual input and output fields in each service.

Using specifications to define service signatures provides the following benefits:

- It reduces the effort required to build each service signature.
- It improves accuracy, because there is less opportunity to introduce a typing error when defining a field name.
- It makes future signature changes easier to implement, because you can make the change in one place (the specification) rather than in each individual service. Any change that you make to the specification is automatically propagated to all services that reference that specification when you save the specification.

Note:
If you want to rename a field in a specifications that is already used by a service, make sure to refactor the field instead of merely renaming. For more information about refactoring, see "Refactoring Variable Names in Elements" on page 54.

Creating a Specification

When you create a specification, you can define the specification content by:

- Manually inserting input and output variables.
- Referencing an IS document type for the input and/or output signature.
- Referencing a specification to use as the entire signature.

Typically, you do not build a specification by referencing another specification. However, it is useful to do this in the situation where you will use the specification with a group of services whose requirements are expected to change (that is, they match an existing specification now but are expected to change at some point in the future). Referencing a specification gives you the convenience of using an existing specification and the flexibility to change the specification for only that single group of services in the future.

To create a specification

1. In Designer: **File > New > Specification**
2. In the New Specification dialog box, select the folder in which you want to save the specification.
3. In the **Element Name** field, type a name for the specification using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
4. Click **Finish**.

5. To define the specification content by referencing another specification, in the **Specification Reference** field in the Input/Output tab, type the fully qualified name of the specification, or click ... to select it from a list.

6. To use an IS document type to define the input content for a specification, in the **Input** field, type the fully qualified name of the IS document type or click ... to select it from a list.

7. To use an IS document type to define the output content for a specification, in the **Output** field, type the fully qualified name of the IS document type or click ... to select it from a list.

8. To define the specification by inserting variables manually, do the following for each variable that you want to add:

   a. In the Palette view, select the type of variable you want to define and drag it to the Input or Output side of the Input/Output tab.

      If the Palette view is not visible, display it by clicking on the right side of the specification editor.

   b. Type a name for the variable and press ENTER.

   c. With the variable selected, set variable properties and apply constraints using the Properties view.

   d. If the variable is a document or document list, repeat steps a–c to define and set the properties and constraints for each of its members. Use to indent each member beneath the document or document list variable.

9. Optionally, enter comments or notes for the specification in the **Comments** tab.

10. Click **File > Save**.

30 Working with Variables

- Creating a Document Reference or a Document Reference List Variable 604
- Using XML Namespaces and Namespace Prefixes with Variables 605
- Assigning Display Types to String Variables 607
- About Variable Constraints 607

A variable can be a String, String list, String table, document, document list, document reference, document reference list, Object, or Object list. Variables are used to declare the expected content and structure of service signatures, document contents, and pipeline contents. In addition to specifying the name and data type of a variable, you can set properties that specify an XML namespace, indicate whether the variable is required at runtime, and indicate whether the variable can be null at runtime.

Select a variable in the editor to set general properties and constraints for the variable. Specific properties in the Properties view are enabled or disabled, depending on the type of variable you have selected.

Note:
Be careful when renaming variables. If a variable is used in pipeline mapping, renaming the variable might result in the pipeline containing a variable with the new name and a variable with the old name. In this situation, the variable with the old name would retain the pipeline mapping. To avoid this, refactor the variable instead. For more information about refactoring, see "Refactoring Variable Names in Elements" on page 54.

Creating a Document Reference or a Document Reference List Variable

You can use an IS document type (including a publishable document type) to build a document reference or document reference list field. By referencing an IS document type instead of creating a new one, you can reduce the time required to create fields and maintain better consistency for field names. You might find referencing fields especially useful for information that is repeated over and over again, such as address information.

Keep the following points in mind when using IS document types to create document references or document reference lists.

- If you are creating a document reference or document reference list field based on an IS document type, you cannot directly add, delete, or modify its members in the Input/Output tab. To edit the referenced document or document list, select it in the Package Navigator view, lock it, and then edit its fields in the editor.
- You can also add a document reference by dragging an IS document type from the Package Navigator view to the Document Type editor.

To use a document type to build a document reference or document reference list

1. In Package Navigator view, open the document type that you want to reference.
2. On the editor palette, click and drag one of the following into the editor window:

Click and drag...	To...
Document Reference	Create a document field based on an IS document type.

Click and drag...	To...
Document Reference List	Create a document list field based on an IS document type.

3. In the **Element Name** field, type the fully qualified name of the IS document type or select it from the list.

4. Click **OK**.

5. Type the name of the field.

6. Click **File > Save**.

Using XML Namespaces and Namespace Prefixes with Variables

You can assign an XML namespace and prefix to a variable by specifying a URI for the **XML namespace** property and by using the following format for the variable name:

prefix:variableName

This creates a QName for the variable in which portions of the variable name are used as the prefix and local name and the value of the **XML namespace** property is the namespace name. For example, suppose that a variable is named *eg:account* and the **XML namespace** property is set to http://www.example.com. The prefix is *eg*, the localname is *account* and the namespace name is http://www.example.com.

When generating XML Schema definitions and WSDL documents, Integration Server uses the value of the **XML Namespace** property along with the variable name (the prefix and local name) to create a qualified name (QName) for the variable.

Integration Server also uses namespace prefixes assigned to variables in an IS document type when converting an XML document or XML node to a document (IData object). Integration Server obtains the namespace prefix to be used in the resulting document (IData) from the variable name in the document type rather than relying on the prefix present in the actual XML node. For example, suppose that Integration Server converts an XML node to a document (IData) using a document type to specify the structure to be imposed on the resulting document (IData). The document type contains a field named *eg:account* for which the **XML Namespace** property is set to www.example.com. The XML node contains an element named otherPrefix:account in which otherPrefix is defined to represent the namespace http://www.example.com. The resulting document (IData) contains a field named *eg:account* based on the document type field instead of *otherPrefix:account* as it appears in the original XML node.

Note: Integration Server automatically assigns an XML namespace to a variable when it creates a provider web service descriptor WSDL or a consumer web service descriptor from an existing WSDL. Integration Server also assigns an XML namespace from a schema when it creates a document type from an existing XML schema definition.

Guidelines for Using XML Namespaces and Prefixes with Web Service Descriptors

Use the following guidelines for assigning XML namespaces and prefixes to variables that will be used with web service descriptors:

- If an IS service signature contains a variable name in the format *prefix*:*localName*, you must specify a URI as the value of the **XML namespace** property for that variable.
- If an IS service is exposed as an RPC/Literal or RPC/Encoded web service operation, then the top-level field name in the service signature should not be in the format *prefix*:*localName* and should not be associated with an XML namespace.
- If an IS service is exposed as a Document/Literal web service operation, then any field name in the service signature can be in the format *prefix*:*localName*, but it must also be associated with an XML namespace.

Assigning XML Namespaces and Prefixes to Variables

Use the **XML Namespace** property to assign a namespace name to a variable. If you intend to use the variable with a web service descriptor (for example, in the signature of a service used as an operation or in an IS document type used for a header or fault), review the information described in "Guidelines for Using XML Namespaces and Prefixes with Web Service Descriptors" on page 606 before assigning XML namespaces and namespace prefixes to a variable.

Keep the following points in mind when assigning XML namespaces and prefixes to a variable:

- The variable name must be in the format: *prefix:variableName*
- You must specify a URI in the **XML namespace** property.
- Do not use the same prefix for different namespaces in the same document type, input signature, or output signature.

To assign an XML namespace and prefix to a variable

1. In the editor, select the variable to which you want to assign an XML namespace.
2. In the Properties view, click the **XML namespace** browse button (...) and then enter a value for the XML namespace.
3. To assign a prefix to a variable, rename the variable to use the following format in which the namespace prefix precedes the variable name: `prefix:variableName`.
4. Click **OK**.
5. Click **File > Save**.

Assigning Display Types to String Variables

Assign a string display type to the string variable to define how you want to input data for the field.

To assign a string display type to a String variable

1. In the editor, select the String variable to which you want to assign properties.
2. In the Properties view, under **General**, assign one of the following values to the **String display type** field:

Select...	If you want the input...
Text Field (default)	Entered in a text field
Password	Entered as a password, with asterisks reflected instead of characters
Large Editor	Entered in a large text area instead of a text field. This is useful if you expect a large amount of text as input for the field, or you need to have TAB or new line characters in the input
Pick List	To be limited to a predefined list of values Next to **Pick list choices**, click … to define the values that will appear as choices when Designer prompts for input at run time.

Note:
These options are not available for Objects and Object lists.

3. Click **File > Save**.

About Variable Constraints

In pipeline, document, and input/output validation, the blueprint used for data validation requires *constraints* to be applied to its variables. Constraints are the restrictions on the structure or content of variables. You can apply the following types of constraints to variables:

- **Structural** constraints specify the existence and structure of variables at run time. For example, if the flow service in which you are validating the pipeline processes a purchase order, you might want to check that values for the *purchaseOrderNumber*, *accountNumber*, and *customerName* variables exist at run time.

  For document and document list variables, you can also specify the structure of the variable; that is, you can specify what variables can be contained in the document (IData object) or document list (IData[ ] object) at run time. For example, you could specify that the *lineItem* document variable must contain the child variables *itemNumber*, *quantity*, *size*, *color*, and *unitPrice*.

You could also specify that the *lineItem* document can optionally contain the child variable *specialInstructions*.

- **Content** constraints describe the data type for a variable and the possible values for the variable at run time. You can apply content constraints to String, String list, String table, Object, and Object list variables.

  When you specify a content constraint for a String, String list or String table variable, you can also apply a *constraining facet* to the content. A constraining facet places limitations on the content (data type). For example, for a String variable named *itemQuantity*, you might apply a content constraint that requires the value to be an integer. You could then apply constraining facets that limit the content of *itemQuantity* to a value between 1 and 100.

  You can use simple types from an IS schema as content constraints for String, String list, or String table variables.

For pipeline and document validation, the IS document type used as the blueprint needs to have constraints applied to its variables. For input/output validation, constraints need to be applied to the declared input and output parameters of the service being validated. For more information about data validation, see "Performing Data Validation" on page 293.

Note:
When you create an IS document type from an XML Schema or a DTD, the constraints applied to the elements and attributes in the original document are preserved in the new IS document type. For more information about creating IS document types, see "Creating an IS Document Type" on page 516.

Considerations for Object Constraints

Constraints for Object and Object list variables correspond to Java classes, whereas constraints for String variables correspond to simple types in XML schemas. When you apply constraints to Objects and perform validation, the data is validated as being of the specified Java class. For details on the Java classes you can apply to an Object or Object list, see "Java Classes for Objects" on page 1085.

A constrained Object is validated when one of the following occur:

- A service runs with the **Validate input** or **Validate output** check box selected on the Input/Output tab.
- A service runs via the INVOKE step in a flow service with the **Validate input** or **Validate output** properties set on the INVOKE step.
- The pub.schema:validate service runs.
- A document is published. When this occurs, the contents of the document are validated against the specified document type.
- During debugging, when you enter values for a constrained Object or Object list in the Input dialog box.

- When you assign a value to an Object or Object list variable in the Pipeline view using on the toolbar.

Applying Constraints to a Variable

Keep the following points in mind when applying constraints to a variable:

- The **Required** property appears for variables in document types if one or more of the following are true:
  - The document type was created using a version of Integration Server prior to version 8.2.
  - The document type was created using Developer.
  - The **Model type** property of the document type is Unordered.
- The **Min occurs**, **Max occurs**, and **Model type** variable properties are display-only. These properties appear only for a variable in an IS document type with a **Model type** property value of Sequence, Choice, or All.
- Variables defined in an IS document type with a **Model type** property value other than "Unordered" cannot be modified.

To apply constraints to a variable

1. Select the variable to which you want to apply constraints.

   You can apply constraints to variables in IS document types, variables in a specification, and variables declared on the Input/Output tab.

2. In the Properties view, under **Constraints**, define the following properties:

Property	Description	
Required	Specifies whether the selected variable is required to exist at run time.	
	Select...	**To...**
	True	Require the selected variable to exist at run time.
	False	Make the existence of the variable optional at run time.
Allow null	Specifies whether a variable is allowed to have a null value.	
	Select...	**To...**
	True	Allow the variable to have a null value.
	False	Have the validation engine generate an error when the variable has a null value.

Property	Description
Allow unspecified fields	If the variable is a document or document list, specifies whether the variable can contain undeclared child variables.

Select...	To...
True	Allow the variable to contain undeclared child variables.
False	Treat any child variables that exist in the pipeline but do not appear in the declared document field as errors at run time.

Note:
The state of the **Allow unspecified fields** property determines whether the document is open or closed. An *open* document permits undeclared fields (variables) to exist at run time. A *closed* document does not allow undeclared fields to exist at run time. Integration Server considers a document to be open if the **Allow unspecified fields** property is set to **True** and considers a document to be closed if the **Allow unspecified fields** property is set to **False**.

3. If the selected variable is a String, String list, or String table, and you want to specify content constraints for the variable, click ... and then do one of the following:

   - If you want to use a content type that corresponds to a built-in simple type in XML Schema, in the **Content type** list, select the type for the variable contents. To apply the selected type to the variable, click **OK**.

     If you want to customize the content type by changing the constraining facets applied to the type, see "Customizing a String Content Type" on page 611.

   - If you want to use a simple type from an IS schema as the content constraint, click **Browse**. In the Browse dialog box, select the IS schema containing the simple type you want to apply. Then, select the simple type you want to apply to the variable. To apply the selected type to the variable, click **OK**.

     Note:
     A content type corresponds to a simple type from an XML Schema definition. All of the choices in the **Content type** list correspond to simple types defined in *XML Schema Part 2: Datatypes*.

4. If the selected variable is an Object or Object list, for the **Java wrapper type** property, select the Java class for the variable contents. If you do not want to apply a Java class or if the Java class is not listed, select **UNKNOWN**.

   For more information about supported Java classes for Objects and Object lists, see "Java Classes for Objects" on page 1085.

5. Repeat this procedure for each variable to which you want to apply constraints in the IS document type, specification, service input, or service output.

6. Click **File > Save**.

Customizing a String Content Type

Instead of applying an existing content type or simple type to a String, String list, or String table variable, you can customize an existing type and apply the new, modified type to a variable. You customize a content type or simple type by changing the constraining facets applied to the type.

When you customize a type, you actually create a new content type. Designer saves the changes as a new content type named *contentType*_customized. For example, if you customize the string content type, Designer saves the new content type as string_customized.

When customizing a content type, keep the following points in mind:

- When you edit the constraining facets, you can only make the constraining facet values more restrictive. The constraining facets cannot become less restrictive.
- The constraining facets you can specify depend on the content type. Note that content types and constraining facets correspond to datatypes and constraining facets defined in XML Schema. For more information about constraining facets for a datatype, see the specification *XML Schema Part 2: Datatypes* (http://www.w3.org/TR/xmlschema-2/).
- The customized content type applies only to the selected variable. To make changes that affect all variables to which the content type is applied, edit the content type in the IS schema. (String content types are simple types from IS schemas.) For more information about editing simple type definitions, see "About Editing Simple Type Definitions" on page 625.

To customize a content type

1. Select the variable to which you want to apply a customized content type.
2. In the **Constraints** category on the Properties view, click the **Content type** browse button (...) and then do one of the following to select the content type you want to customize:
   - In the **Content type** list, select the content type you want to customize.
   - If you want to customize a simple type from an IS schema, click **Browse**. In the Browse dialog box, select the IS schema containing the simple type. Then, select the simple type you want to customize and apply to the variable. Click **OK**.
3. Click **Customize**. Designer makes the constraining facet fields below the **Content type** list available for data entry (that is, changes the background of the constraining facet fields from grey to white). Designer changes the name of the content type to `contentType`_customized.
4. In the fields below the **Content type** list, specify the constraining facet values you want to apply to the content type.
5. Click **OK**. Designer saves the changes as a new content type named `contentType`_customized.

Note:
The constraining facets displayed below the **Content type** list depend on the primitive type from which the simple type is derived. *Primitive types* are the basic data types from which all other data types are derived. For example, if the primitive type is string, Designer displays the constraining facets **enumeration**, **length**, **minLength**, **maxLength**, and **pattern**. For more information about primitive types, refer to *XML Schema Part 2: Datatypes* at http://www.w3.org/TR/xmlschema-2/.

Viewing the Constraints Applied to Variables

Designer displays small symbols next to a variable icon to indicate the constraints applied to the variable. Designer displays variables in the following ways:

Variable	Constraint status	Variable Properties
	Required field.	The **Required** property is set to True.
	Optional field.	The **Required** property is set to False.
	Required field with content type constraint.	The **Content type** property specifies an IS schema or XML schema.
	Optional field with content type constraint	The **Required** property is set to False and the Content type property specifies an IS schema or XML schema.
	Required field with default value.	The **Required** property is set to True. The variable has a default value, but you can override this default value with any other valid values while executing the service or mapping the variables.
	Required field with fixed default value of null.	The **Required** property is set to True. You cannot override the default null value assigned to the variable by mapping it to another variable or by assigning any input values to this variable during service execution. **Note:** By default, Designer hides the variables with this constraint. To display these variables in the content and structure of service signatures, document and pipeline contents, and in the Run Configurations, Enter Input for *serviceName*, and Enter Input for *variableName* dialog boxes, select the **Show variables with fixed values**

Variable	Constraint status	Variable Properties
		property on the Service Development Preferences page.
	Required field with fixed default value.	The **Required** property is set to True. You cannot override the default value assigned to the variable by mapping it to another variable or by assigning any input values to this variable during service execution. **Note:** By default, Designer hides the variables with this constraint. To display these variables in the content and structure of service signatures, document and pipeline contents, and in the Run Configurations, Enter Input for *serviceName*, and Enter Input for *variableName* dialog boxes, select the **Show variables with fixed values** property on the Service Development Preferences page.

Note: Designer displays the ‡ symbol next to String, String List, and String table variables with a content type constraint only. Designer does not display the ‡ symbol next to Object and Object list variables with a specified Java class constraint. Object and Object lists with an applied Java class constraint have a unique icon. For more information about icons for constrained Objects, see "Java Classes for Objects" on page 1085.

31 Working with Schemas

- What Does an IS Schema Look Like? 616
- Creating an IS Schema 621
- About Editing Simple Type Definitions 625
- About Schema Domains 627

An IS schema is a “free-standing” element in Package Navigator view that acts as the blueprint or model against which you validate an XML document. The IS schema provides a formal description of the structure and content for a valid instance document (the XML document). The formal description is created through the specification of constraints. An IS schema can contain the following types of constraints:

- **Structural constraints** in an IS schema describe the elements, attributes, and types that appear in a valid instance document. For example, an IS schema for a purchase order might specify that a valid `<lineItem>` element must consist of the `<itemNumber>`, `<size>`, `<color>`, `<quantity>`, and `<unitPrice>` elements in that order.
- **Content constraints** in an IS schema describe the type of information that elements and attributes can contain in a valid instance document. For example, the `<quantity>` element might be required to contain a value that is a positive integer.

During data validation, Designer compares the elements and attributes in the instance document with the structural and content constraints described for those elements and attributes in the IS schema. Designer considers the instance document to be valid when it complies with the structural and content constraints described in the IS schema.

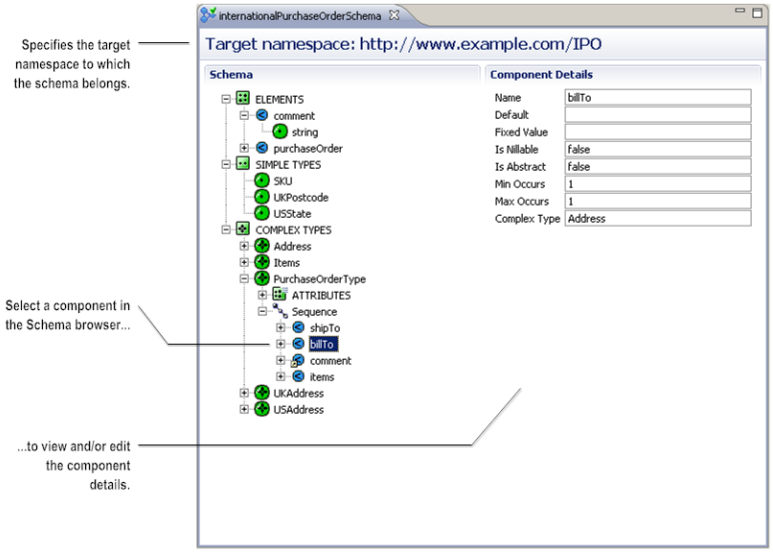
You can create IS schemas from an XML schema, a DTD (Document Type Definition), or an XML document that references an existing DTD.

What Does an IS Schema Look Like?

The appearance and content of an IS schema depends on whether you generate an IS schema from an XML schema or a DTD. For example, if you create an IS schema from an XML schema, the resulting IS schema displays type definitions, element declarations, and attribute declarations. If you create an IS schema from a DTD, the resulting IS schema displays element type declarations.

When you select an IS schema in Package Navigator view, Designer displays the contents of the IS schema in the editor. The schema editor is divided into two areas: the Schema browser on the left and the Component Details on the right. Above these areas, the editor identifies the target namespace for the IS schema.

Schema editor



Schema Browser

The Schema browser displays the components of an IS schema in a format that mirrors the structure and content of the source file. The Schema browser groups the global element declarations, attribute declarations, simple type definitions, and complex type definitions from the source file under the top-level headings ELEMENTS, ATTRIBUTES, SIMPLE TYPES, and COMPLEX TYPES. For example, the ELEMENTS heading contains all of the global element declarations from the XML schema or the DTD.

If the source file does not contain one of these global components, the corresponding heading is absent. For example, if you create an IS schema from an XML schema that does not contain any global attribute declarations, the Schema browser does not display the ATTRIBUTES heading. An IS schema created from a DTD never displays the SIMPLE TYPES or COMPLEX TYPES headings because DTDs do not contain type definitions.

Note:
A DTD does contain attribute declarations. However, the Schema browser does not display the ATTRIBUTES heading for IS schemas generated from DTDs. This is because an attribute declaration in a DTD associates the attribute with an element type. Accordingly, the Schema browser displays attributes as children of the element type declaration to which they are assigned.

The Schema browser uses unique symbols to represent the components of the IS schema. Each of these symbols relates to a component of an XML schema or a DTD. The following table identifies the symbol for each component that can appear in an IS schema.

Note:
In the following table, global refers to elements, attributes, and types declared or defined as immediate children of the `<schema>` element in an XML schema. All element type declarations in a DTD are considered global declarations.

Symbol	Description
	ELEMENTS. Used to group together the global element declarations in an IS schema. This symbol and category name do not correspond to a component in an XML Schema definition or a DTD.
	Element declaration. An element declaration associates an element name with a type definition. This symbol corresponds to the `<element>` declaration in an XML schema definition and the ELEMENT declaration in a DTD.
	Element reference. An element reference is a reference from an element declaration in a content specification to a globally declared element. In an IS schema generated from an XML schema, this symbol corresponds to the `ref="globalElementName"` attribute in an `<element>` declaration. In an IS schema generated from DTD, this symbol appears next to an element that is a child of another element. The parent element has only element content.
	Any element declaration. In XML schema definition, an `<any>` element declaration is a wildcard declaration used as a placeholder for one or more undeclared elements in an instance document. In a DTD, an element declared to be of type ANY can contain any well-formed XML. This symbol corresponds to an element declared to be of type ANY. Because an `<any>` element declaration does not have a name, the Schema browser uses 'Any' as the name of the element.
	ATTRIBUTES. Used to group together the global attribute declarations in an IS schema. This symbol and category name do not correspond to a component in an XML Schema definition or a DTD.
	Attribute declaration. An attribute declaration associates an attribute name with a simple type definition. This symbol corresponds to the XML schema `<attribute>` declaration or the attribute in a DTD ATTLIST declaration.
	Attribute reference. An attribute reference is a reference from a complex type definition to a globally declared attribute. This symbol corresponds to the `ref="globalAttributeName"` attribute in an attribute declaration. DTDs do not have attribute references. Consequently, attribute references do not appear in IS schemas generated from DTDs.
	Any attribute declaration. An any attribute declaration is a wildcard declaration used as a placeholder for undeclared attributes in an

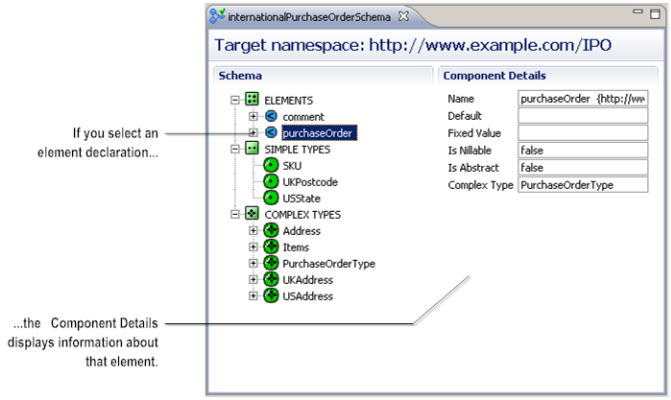
Symbol	Description
	instance document. This symbol corresponds to the `<anyAttribute>` declaration in an XML schema definition. Because an `<anyAttribute>` declaration does not specify an attribute name, the Schema browser uses 'Any' as the name of the attribute.
	SIMPLE TYPES. Used to group together the global simple type definitions in an IS schema. This symbol and category name do not correspond to a component in an XML Schema definition or a DTD.
	Simple type definition. A simple type definition specifies the data type for a text-only element or an attribute. Unlike complex type definitions, simple type definitions cannot carry attributes. This symbol corresponds to the `<simpleType>` element in an XML schema definition. If the simple type definition is unnamed (an anonymous type), the Schema browser displays 'Anonymous' as the name of the simple type definition.
	COMPLEX TYPES. Used to group together the global complex type definitions in an IS schema. This symbol and category name do not correspond to a component in an XML Schema definition or a DTD.
	Complex type definition. A complex type definition defines the structure and content for elements of complex type. (Elements of complex type can contain child elements and carry attributes.) This symbol corresponds to the `<complexType>` element in an XML schema definition. If the complex type definition is unnamed (an anonymous type), the Schema browser displays 'Anonymous' as the name of the complex type definition.
	Sequence content model. A sequence content model specifies that the child elements in the instance document must appear in the same order in which they are declared in the content model. This symbol corresponds to the `<sequence>` compositor in an XML schema or a sequence list in an element type declaration in a DTD.
	Choice content model. A choice content model specifies that only one of the child elements in the content model can appear in the instance document. This symbol corresponds to the `<choice>` compositor in an XML schema definition or a choice list in a DTD element type declaration.
	All content model. An all content model specifies that child elements can appear once, or not at all, and in any order in the instance document. This symbol corresponds to the `<all>` compositor in an XML schema definition.

Symbol	Description
	Mixed content. Elements that contain mixed content allow character data to be interspersed with child elements. This symbol corresponds to the `mixed="true"` attribute in an XML schema complex type definition or a DTD element list in which the first item is `#PCDATA`.
	Empty content. In an XML schema, an element has empty content when its associated complex type definition does not contain any element declarations. An element with empty content may still carry attributes. In a DTD, an element has empty content when it is declared to be of type `EMPTY`.

Component Details

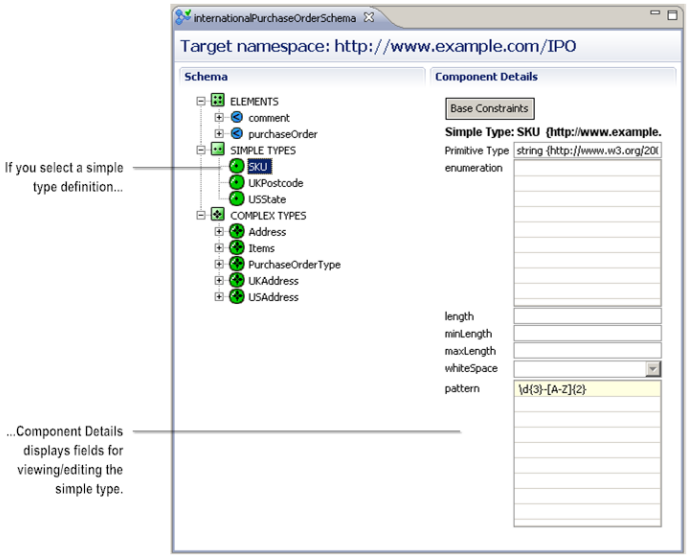
The Component Details area displays information that you use to examine and edit the selected component in the Schema browser. The contents of Component Details varies depending on what component you select. For example, when you select a globally declared element of complex type, the Component Details looks like the following:

Schema editor for an element declaration



When you select a simple type definition, the Component Details looks like the following:

Schema editor for a simple type definition



Creating an IS Schema

You can create IS schemas from XML schema definitions, DTDs, and XML documents that reference an existing DTD. The resulting IS schema contains all of the defined types, declared elements, and declared attributes from the source file.

You can create an IS schema from an XML Schema definition in CentraSite. To do so, Designer must be configured to connect to CentraSite.

To create an IS schema

1. In the Package Navigator view of the Service Development perspective, click **File > New > Schema**.

2. In the New Schema dialog box, select the folder in which you want to save the IS document type.

3. In the **Element Name** field, type a name for the IS document type using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

4. Click **Next**.

5. On the Select the Source Type panel, do one of the following:

Select...	To...
XML	Create an IS schema based on an existing DTD referenced by an XML document.

Select...	To...
	Note: You can create an IS schema from an XML document only if the XML document references an existing DTD.
DTD	Create an IS schema based on a DTD.
XML Schema	Create an IS schema based on an XML schema definition.

6. On the Select a Source Location panel, under **Source location**, do one of the following to specify the source file for the IS schema:

   - To use an XML schema definition in CentraSite as the source, select **CentraSite**.
   - To use an XML document, DTD, or XML schema definition that resides on the Internet as the source, select **File/URL**. Then, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
   - To use an XML document, DTD, or XML Schema definition that resides on your local file system as the source, select **File/URL**. Then, type in the path and file name, or click the **Browse** button to navigate to and select the file.

7. Click **Next**.

8. If you selected CentraSite as the source, under **Select XML Schema from CentraSite**, select the XML Schema definition in CentraSite that you want to use to create the IS schema. Click **Next**.

   If Designer is not configured to connect to CentraSite, Designer displays the **CentraSite > Connections** preference page and prompts you to configure a connection to CentraSite.

   Note:
   You can also create an IS schema from an XML Schema definition asset in CentraSite by dragging and dropping the schema asset from the Registry Explorer view into Package Navigator view.

9. If the source file is an XML Schema definition, on the Select Schema Domain panel, under **Schema domain**, specify the schema domain to which any generated IS schemas will belong. Do one of the following:

   - To add the IS schema to the default schema domain, select **Use default schema domain**.
   - To add the IS schemas to a specified schema domain, select **Use specified schema domain** and provide the name of the schema domain in the text box. A valid schema domain name is any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.

   For more information about schema domains, see "About Schema Domains" on page 627.

10. If the source file is an XML Schema definition and you want Integration Server to use the Xerces Java parser to validate the XML Schema definition, select the **Validate schema using Xerces** check box.

    Note: Integration Server automatically uses an internal schema parser to validate the XML Schema definition. However, the Xerces Java parser provides stricter validation than theIntegration Server internal schema parser. As a result, some schemas that the internal schema parser considers to be valid might be considered invalid by the Xerces Java parser.

11. Click **Finish**.

    Designer generates the IS schema using the document you specified and displays it in the editor

Notes:

- Integration Server uses the internal schema parser to validate an XML schema definition. If you selected the **Validate schema using Xerces** check box, Integration Server also uses the Xerces Java parser to validate the XML Schema definition. With either parser, if the XML Schema does not conform syntactically to the schema for XML Schemas defined in *XML Schema Part 1: Structures* (which is located at http://www.w3.org/TR/xmlschema-1), Integration Server does not create an IS schema. Instead, Integration Server displays an error message that lists the number, title, location, and description of the validation errors within the XML Schema definition. If only warnings occur, Designer generates the IS schema.

  Note: Integration Server uses Xerces Java parser version J-2.11.0. Limitations for this version are listed at http://xerces.apache.org/xerces2-j/xml-schema.html.

- When validating XML schema definitions, Integration Server uses the Perl5 regular expression compiler instead of the XML regular expression syntax defined by the World Wide Web Consortium for the XML Schema standard. As a result, in XML schema definitions consumed by Integration Server, the pattern constraining facet must use valid Perl regular expression syntax. If the supplied pattern does not use proper Perl regular expression syntax, Integration Server considers the pattern to be invalid.

  Note:
  If the watt.core.datatype.usejavaregex configuration parameter is set to true, Integration Server uses the Java regular expression compiler instead of the Perl5 regular expression compiler. When the parameter is true, the pattern constraining facet in XML schema definitions must use valid syntax as defined by the Java regular expression.

- Integration Server does not create IS schemas from an XML schema definition (XSD) if the XSD contains a type definition derived by extension and that type definition contains a direct or indirect reference to itself. If Integration Server encounters a type definition that contains a recursive extension while creating an IS schema from an XSD, Integration Server throws a StackOverflowError and does not continue creating the IS schema.

- You might receive errors or warnings when creating an IS schema from a DTD. If one or more errors occur, Designer does not generate an IS schema. If only warnings occur, Designer generates the IS schema.

- When creating an IS schema from an XML Schema definition that imports multiple schemas from the same target namespace, Integration Server throws Xerces validation errors indicating that the element declaration, attribute declaration, or type definition cannot be found. The Xerces Java parser honors the first <import> and ignores the others. To work around this issue, you can do one of the following:
  - Combine the schemas from the same target namespace into a single XML Schema definition. Then change the XML schema definition to import the merged schema only.
  - When creating the IS schema, clear the **Validate schema using Xerces** check box to disable schema validation by the Xerces Java parser. When generating the IS schema, Integration Server will not use the Xerces Java parser to validate the schemas associated with the XML Schema definition.

Creating an IS Schema from XML Schemas that Reference Other Schemas

A schema author can insert the elements, attributes, and type definitions from another schema into the schema they are creating. A schema author might do this to break up a large XML schema into several small, more reusable XML schemas. When you generate an IS schema from an XML schema that references another schema, Integration Server either includes all of the schema components in a single IS schema, creates multiple IS schemas, or creates no IS schema at all. The behavior of the Integration Server depends on the mechanism the source XML schema uses to reference the other schema. The following mechanisms can be used to reference an external schema:

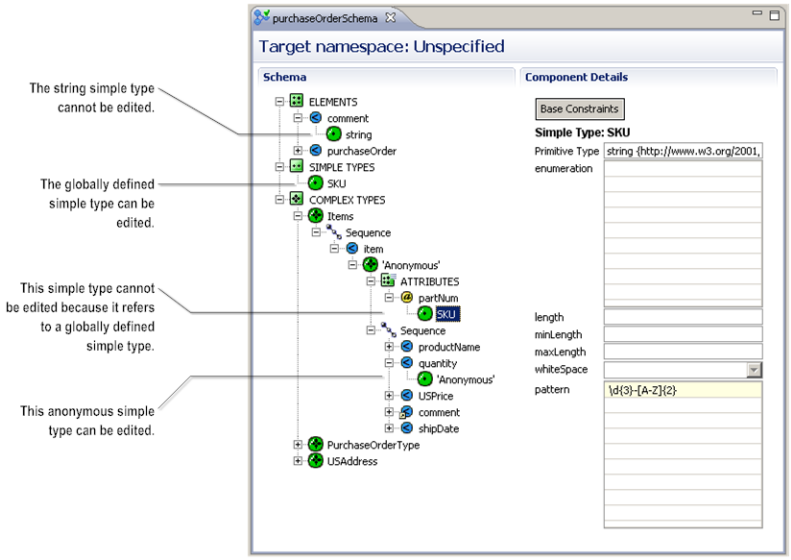
- **Include.** When you generate an IS schema from an XML schema that uses `<include>` to include the contents of an external schema in the same namespace, the resulting IS schema contains all of the defined types, declared elements, and declared attributes from the source schema and the external schema. If the including, or root, XML schema references an external schema that does not contain a target namespace declaration, the external schema assumes the target namespace of the root schema.
- **Import.** When you generate an IS schema from an XML schema that contains an `<import>` element to import the contents of an external schema in a different namespace, Integration Server creates one IS schema per namespace. For example, if the source XML schema imports two XML schemas from the same namespace, Integration Server creates an IS schema for the source XML schema and then a second IS schema that includes the components from the two imported XML schemas. Integration Server assigns each imported schema the name that you specify and appends an underscore and a number to each name. For example, if you create an IS schema named “mySchema” from mySchema.xsd, Integration Server generates an IS schema named “mySchema_2” for the imported XML schema.
- **Redefine.** Schema authors can also use `<redefine>` to include and then redefine type definitions, model groups, and attribute groups from an external XML schema in the same namespace. Integration Server creates one IS schema per namespace.

About Editing Simple Type Definitions

You can modify a simple type definition in an IS schema without editing the source XML schema definition. You edit a simple type by adding or changing the value of one or more constraining facets applied to the simple type. For example, you can modify a simple type by adding an enumerated value, a pattern constraint, or changing the length constraint. Editing the simple type through Designer is an alternative to editing the source XML schema definition and regenerating the IS schema.

You can edit any of the globally defined simple types (those that appear under the SIMPLE TYPES heading) or anonymous simple types (those defined as part of an element or attribute declaration.) A named simple type that appears as a child of an element or attribute in the Schema browser cannot be edited. (These simple types are global simple types used to define the element or attribute they appear under.) The following illustration identifies the simple type definitions that you can and cannot edit.

Editable simple type definitions in an IS schema



You can edit any of the constraining facet values that appear in the Component Details when you select an editable simple type definition in the Schema browser. The constraining facets displayed in the Component Details depend on the primitive type from which the simple type was derived. For example, if the simple type definition is derived from string, the Component Details displays the **enumeration**, **length**, **minLength**, **maxLength**, **pattern**, and **whiteSpace** facets. The Component Details only displays constraining facet values set in the simple type definition. It does not display constraining facet values the simple type definition inherited from the simple types from which it was derived.

You can view the constraining facet values set in the type definitions from which a simple type was derived by clicking **Base Constraints**. Base constraints are the constraining facet values set in all the type definitions from which a simple type is derived—from the primitive type to the immediate parent type. These constraint values represent the cumulative facet values for the simple type.

When you edit the constraining facets for a simple type definition, you can only make the constraining facets more restrictive. The applied constraining facets cannot become less restrictive. For example, if the **length** value is applied to the simple type, the **maxLength** or **minLength** values cannot be set because the **maxLength** and **minLength** facets are less restrictive than **length**.

Tip:
You can create a custom simple type to apply to a field as a content type constraint. For information about creating a custom simple type, see "Customizing a String Content Type" on page 611. For information about applying constraints to fields, see "Applying Constraints to a Variable" on page 609.

Editing a Simple Type Definition

When modifying a simple type definition, keep the following points in mind:

- Changes to a simple type definition affect the elements and attributes for which the simple type is the defined type. For example, if the attribute `partNum` is defined to be of type `SKU`, changes to the `SKU` simple type definition affect the `partNum` attribute.
- Changes to a global simple type definition in an IS schema do not affect simple types derived from the global simple type; that is, the changes are not propagated to the derived types in the IS schema.
- Simple types in an IS schema can be used as content constraints for fields in pipeline validation. Consequently, changes to a simple type also affect every field to which the simple type is applied as a content constraint.
- Changes to a simple type definition are saved in the IS schema. If you regenerate the IS schema from the XML schema definition, your changes will be overwritten.
- When you edit the constraining facets applied to a simple type definition, you can only make the constraining facet values more restrictive. The constraining facets cannot become less restrictive.
- If you want to edit complex type definitions, attribute declarations, element declarations, or the structure of the schema, you need to edit the XML schema and then regenerate the IS schema.
- For an IS schema from a source file such as an XML schema definition or a WSDL document Designer displays the location of the source file in the **Source URI** property. Designer also sets the **Linked to source** property to true which prevents any editing of the simple type definitions in the IS schema. To edit the simple type definitions, you first need to make the IS schema editable by breaking the link to the source. For information about allowing editing of elements derived from a source, see "Allowing Editing of Derived Elements" on page 47. However, Software AG does not recommend editing the contents of IS schemas created from WSDL documents.
- For more information about constraining facets, see the W3C specification *XML Schema Part 2: Datatypes*.

To edit a simple type definition

1. In the Package Navigator view of the Service Development perspective, open the IS schema that contains the simple type that you want to edit.

2. In the Schema Browser, select the simple type definition that you want to edit.

3. In Component Details, specify the constraining facets that you want to apply to the simple type definition.

4. Click **File > Save**.

About Schema Domains

Each IS schema on Integration Server belongs to a *schema domain*. A schema domain is a named collection of IS schemas.

Within each schema domain, the schema components such as element declarations, attribute declarations, type definitions for an XML namespaces must be unique. However, more than one schema domain can contain the same schema components for an XML namespace. This allows Integration Server to maintain multiple IS schemas for the same XML namespace.

If you want to maintain alternate definitions of schema components for an XML namespaces, assign the IS schemas that contain those components to different schema domains.

Integration Server contains a default schema domain in which any IS schemas created prior to version 8.0 reside. You can place new schemas in the default schema domain or you can specify a different schema domain. A schema domain name is any string that is a valid name of an element in Integration Server.

When Integration Server consumes a WSDL document to create a web service descriptor, Integration Server places any generated IS schemas in a unique schema domain for that web service descriptor. Integration Server uses a combination of alphanumeric characters as the schema domain name.

When Integration Server creates an IS document type from an e-form template, Integration Server places any generated IS schemas in a unique schema domain whose name is based on the path to and name of the e-form template.

32 Working with JMS Triggers

- About SOAP-JMS Triggers 630
- Overview of Building a Non-Transacted JMS Trigger 631
- Standard JMS Trigger Service Requirements 632
- Creating a JMS Trigger 633
- Managing Destinations and Durable Subscribers on the JMS Provider through Designer 642
- Building Standard JMS Triggers with Multiple Routing Rules 643
- Enabling or Disabling a JMS Trigger 644
- Setting an Acknowledgement Mode 646
- About Join Time-Outs 647
- About Execution Users for JMS Triggers 649
- About Message Processing 649
- Fatal Error Handling for Non-Transacted JMS Triggers 656
- Transient Error Handling for Non-Transacted JMS Triggers 658
- Exactly-Once Processing for JMS Triggers 664
- Debugging a JMS Trigger 666
- Building a Transacted JMS Trigger 668

A JMS trigger subscribes to destinations (queues or topics) on a JMS provider and then specifies how Integration Server processes messages the JMS trigger receives from those destinations. Integration Server and Designer support two types of JMS triggers:

- **Standard JMS triggers** use routing rules to specify which services can process messages received by the trigger. The trigger service in the routing rule receives the entire JMS message as an IData.
- **SOAP- JMS triggers** are used to receive JMS messages that contain SOAP messages. When a SOAP-JMS trigger receives a message, Integration Server extracts the SOAP message from the JMS message and passes the SOAP message to the internal web services stack. The web services stack processes the message according to the web service descriptor specified in the SOAP-JMS request.

  Note:
  WS endpoint triggers are SOAP-JMS triggers. However, WS endpoint triggers can be created and managed using Integration Server Administrator only. For more information about WS endpoint triggers, see *webMethods Integration Server Administrator's Guide*.

Standard JMS triggers and SOAP-JMS triggers can be transacted or non-transacted triggers. The transactionality of a JMS trigger along with the trigger type affect the properties and functionality that can be configured for the trigger.

Note:
Information about using Integration Server for JMS is located in *webMethods Integration Server Administrator's Guide*, *webMethods Service Development Help*, and *Using webMethods Integration Server to Build a Client for JMS*.

- *webMethods Integration Server Administrator's Guide* contains information about how to configure Integration Server to work with a JMS provider, how to create a WS endpoint trigger, and how to manage JMS triggers at run time.
- *webMethods Service Development Help* includes this "Working with JMS Triggers" on page 629 topic which provides procedures for using Designer to create JMS triggers and set JMS trigger properties.
- *Using webMethods Integration Server to Build a Client for JMS* contains information such as how to build services that send and receive JMS messages, how Integration Server works with cluster policies when sending JMS messages, and detailed information regarding how Integration Server performs exactly-once processing. For completeness, *Using webMethods Integration Server to Build a Client for JMS* also includes the "Working with JMS Triggers" on page 629 topic that appears in *webMethods Service Development Help*.

About SOAP-JMS Triggers

A SOAP-JMS trigger is a JMS trigger that receives SOAP over JMS messages and routes the SOAP message to the web services stack for processing. More specifically, the SOAP-JMS trigger receives JMS messages from a destination (queue or topic) on the JMS provider. Note that a SOAP-JMS trigger can specify a message selector which limits the messages the SOAP-JMS trigger receives from that destination. Integration Server extracts the SOAP message and passes it to the internal web services stack for processing. Integration Server also retrieves JMS message properties that it passes to the web services stack, including targetService, soapAction, contentType, and JMSMessageID. These properties specify the web service descriptor and operation for which the

SOAP request is intended. The web services stack then processes the SOAP message according to the web service descriptor (for example, executing request handlers) and invokes the web service operation specified in the SOAP request message.

A SOAP-JMS trigger is associated with one or more provider web service descriptors via a provider web service endpoint alias. The provider web service endpoint alias specifies the SOAP-JMS trigger that receives messages from destinations on the JMS provider. The provider web service endpoint alias is assigned to a JMS binder in a provider web service descriptor. In this way, SOAP-JMS triggers act as listeners for provider web service descriptors.

Note:
Even though a SOAP-JMS trigger is associated with one or more provider web service descriptors, the SOAP-JMS trigger can pass any SOAP-JMS message to the web services stack for processing.

The properties assigned to the SOAP-JMS trigger determine how Integration Server acknowledges the message, provides exactly-once processing, or handles transient or fatal errors.

While SOAP-JMS triggers and standard JMS triggers share many properties and characteristics, some properties available to standard JMS triggers are not available to SOAP-JMS triggers, specifically:

- SOAP-JMS triggers can subscribe to one destination only. Consequently, SOAP-JMS triggers do not have joins. Designer does not display the **Join expires** and **Expire after** properties for a SOAP-JMS trigger.
- SOAP-JMS triggers use web services to process the payload of the JMS message. Designer does not display the Message Routing table for SOAP-JMS triggers.
- SOAP-JMS triggers cannot be used to perform ordered service execution. Standard JMS triggers use multiple routing rules and local filters to perform ordered service execution. Because SOAP-JMS triggers do not use routing rules, SOAP-JMS triggers cannot be used to perform ordered service execution.
- A SOAP-JMS trigger, specifically a connection for a SOAP-JMS trigger, can process only one message at a time. Batch processing is not available for SOAP-JMS triggers. Designer does not display the **Max batch processing** property for SOAP-JMS triggers.
- A transacted SOAP-JMS trigger (one that executes as part of a transaction) has additional requirements and limitations when used with web service descriptors. For more information, see the *Web Services Developer's Guide*.

Overview of Building a Non-Transacted JMS Trigger

Building a JMS trigger is a process that involves the following basic stages.

Stage 1	Create a new JMS trigger on Integration Server. During this stage, you use Designer to create the new JMS trigger on the Integration Server where you will do your development and testing.
Stage 2	Specify a JMS connection alias.

During this stage, you specify the JMS connection alias that Integration Server uses to create connections to the JMS provider. The transaction type of the JMS connection alias determines whether or not the JMS trigger receives and processes messages as part of transaction.

Stage 3 Specify JMS destinations and message selectors.

During this stage, you specify the destinations (queues or topics) on the JMS provider to which the JMS trigger subscribes. That is, the destination is the source of the messages that the JMS trigger consumes. You also specify any message selectors that you want the JMS provider to use to filter the messages it enqueues for the JMS trigger.

Stage 4 Create routing rules (for standard JMS triggers only).

During this stage, you specify the service that Integration Server invokes when the standard JMS trigger receives messages. You can also specify a local filter that Integration Server applies to messages.

Stage 5 Set JMS trigger properties.

During this stage, you determine the type of message processing, the acknowledgement mode, fatal and transient error handling, and exactly-once processing.

Stage 6 Test and debug the JMS trigger.

During this stage, you test and debug the trigger using the tools provided by Integration Server. For more information, see "Debugging a JMS Trigger" on page 666.

Standard JMS Trigger Service Requirements

The service that processes a message received by a standard JMS trigger is called a *trigger service*. Each routing rule in a standard JMS trigger specifies a single trigger service.

Before a JMS trigger can be enabled, the trigger service must already exist on the same Integration Server.

The signature for the trigger service must reference one of the following specifications:

- Use pub.jms:triggerSpec as the specification reference if the trigger service will process one message at a time.
- Use pub.jms:batchTriggerSpec as the specification reference if the trigger service will process multiple messages at one time. That is, the trigger service will receive a batch of messages as input and process all of those messages in a single execution. A trigger that receives and processes a batch of messages is sometimes referred to as a *batch trigger*.

Creating a JMS Trigger

When you create a JMS trigger, keep the following points in mind:

- The JMS connection alias you want Integration Server to use to obtain connections to and receive messages from the JMS provider must already exist. Although a JMS connection alias does not need to be enabled at the time you create the JMS trigger, the JMS connection alias must be enabled for the JMS trigger to execute at run time.

  Note:
  If you want to manage destinations and durable subscribers on a webMethods Broker that is used as a JMS provider, the JMS connection alias must be enabled when you work with the JMS trigger.

- If you use a JNDI provider to store JMS administered objects, the Connection Factories that you want the JMS trigger to use to consume messages must already exist.
- If you use a JNDI provider to store JMS administered objects and the JMS provider is not webMethods Broker, the destinations (queues and topics) from which this JMS trigger will receive messages must already exist.
- If the JMS provider is webMethods Broker, Software AG Universal Messaging, or webMethods Nirvana the destinations (queues and topics) from which the JMS trigger receives messages do not need to exist before you create the JMS trigger. Instead, you can create destinations using the JMS trigger editor. You can also create, modify, and delete durable subscribers via the JMS trigger. For more information, see "Managing Destinations and Durable Subscribers on the JMS Provider through Designer " on page 642.
- If the JMS provider is Universal Messaging, administered objects can be created on demand. If you specify a destination that does not exist in the JNDI namespace and the JMS connection alias used by the JMS trigger is configured to create administered objects on demand, Integration Server creates the destination the first time the JMS trigger starts. For more information about creating administered objects on demand, see *webMethods Integration Server Administrator's Guide*.
- The transaction type of the JMS connection alias determines whether or not the JMS trigger is transacted (that is, it receives and processes messages as part of a transaction). Transacted JMS triggers have slightly different properties and operate differently than non-transacted JMS triggers. For more information about building a transacted JMS trigger, see "Building a Transacted JMS Trigger" on page 668.
- The trigger service that you want to specify in the routing rule must already exist on the same Integration Server on which you create the JMS trigger. For more information, see "Standard JMS Trigger Service Requirements" on page 632.
- A standard JMS trigger can contain multiple routing rules. Each routing rule must have a unique name. For more information about using multiple routing rules, see "Building Standard JMS Triggers with Multiple Routing Rules" on page 643.
- A standard JMS trigger that contains an All (AND) or Only one (XOR) join can only have one routing rule and cannot have a batch processing size (**Max batch messages** property) greater

than 1. A JMS trigger with an Any (Or) join can have multiple routing rules. For more information about batch processing, see "About Batch Processing for Standard JMS Triggers" on page 651.

- Integration Server uses a consumer to receive messages for a JMS trigger. This consumer encapsulates the actual javax.jms.MessageConsumer and javax.jms.Session.

To create a JMS trigger

1. In the Package Navigator view of Designer, click **File > New > JMS Trigger**.
2. In the New JMS Trigger dialog box, select the folder in which you want to save the JMS trigger.
3. In the **Element name** field, type a name for the JMS trigger using any combination of letters, numbers, and/or the underscore character.
4. Click **Finish**.
5. In the **JMS connection alias name** field in the Trigger Settings tab, click .

   Note:
   A transacted JMS connection alias cannot be assigned to a JMS trigger if a cluster policy is applied to the connection factory used by the JMS connection alias.

6. In the Select a JMS connection alias for *triggerName* dialog box, select the JMS connection alias that you want this JMS trigger to use to receive messages from the JMS provider. Click **OK**.

   Designer sets the **Transaction type** property to match the transaction type specified for the JMS connection alias.

   If a JMS connection alias has not yet been configured on Integration Server, Designer displays a message stating the JMS subsystem has not been configured. For information abut creating a JMS connection alias, see *webMethods Integration Server Administrator's Guide*.

7. In the **JMS trigger type** list, select one of the following:

Select	To...
Standard	Create a standard JMS trigger.
SOAP-JMS	Create a SOAP-JMS trigger.

8. Under **JMS destinations and message selectors**, specify the destinations from which the JMS trigger will receive messages. For more information, see "Adding JMS Destinations and Message Selectors to a JMS Trigger" on page 635.

   Note:

For SOAP-JMS triggers, you can specify one destination only.

9. If you selected multiple destinations, select the join type. The join type determines whether Integration Server needs to receive messages from all, any, or only one of destinations to execute the trigger service.

Select...	If you want...
All (AND)	Integration Server to invoke the trigger service when the trigger receives a message from every destination within the join time-out period. The messages must have the same activation.
Any (OR)	Integration Server to invoke the trigger service when the trigger receives a message from any of the specified destinations. This is the default join type. **Note:** Using an Any (OR) join is similar to creating multiple JMS triggers that listen to different destinations. While a JMS trigger with an Any (OR) join will use fewer resources (a single thread will poll each destination for messages), it may cause a decrease in performance (it may take longer for one thread to poll multiple destinations).
Only one (XOR)	Integration Server to invoke the trigger service when it receives a message from any of the specified destinations. For the duration of the join time-out period, the Integration Server discards any messages with the same activation that the trigger receives from the specified destinations.

10. If this is a standard JMS trigger, under **Message routing**, add routing rules. For more information, see "Adding Routing Rules to a Standard JMS Trigger" on page 641.

11. In the Properties view, set properties for the JMS trigger.

12. Enter comments or notes, if any, in the **Comments** tab.

13. Click **File > Save**.

Adding JMS Destinations and Message Selectors to a JMS Trigger

The destination is the queue or topic to which the JMS trigger subscribes on the JMS provider. When a JMS trigger subscribes to a topic, you can also indicate whether Integration Server creates a durable subscriber or a non-durable subscriber for the topic.

To add a JMS destination to a JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger.

2. In the Trigger Settings tab, under **JMS destinations and message selectors**, click .

3. In the **Destination Name** column, do one of the following to specify the destination from which you want the JMS trigger to receive messages.

   - If the JMS connection alias uses JNDI to retrieve administered objects, specify the lookup name of the Destination object.
   - If the JMS connection alias uses the native webMethods API to connect directly to webMethods Broker, specify the provider-specific name of the destination.
   - If the JMS connection alias creates a connection on webMethods Broker, Universal Messaging, or Nirvana, click to select from a list of existing destinations. You can also create a destination and then select it. After you select the destination, click **OK**.

     If the **Order By** mode for the selected destination does not match the existing message processing mode, Designer prompts you to change the processing mode. This situation can occur only when the JMS provider is webMethods Broker.

     For instructions for creating a destination, see "Creating a Destination on the JMS Provider" on page 637.

4. In the **Destination Name** column, in the **Destination Type** column, select the type of destination:

Select...	If...
Queue	The destination is a queue. This is the default.
Topic	The destination is a topic.
Topic (Durable Subscriber)	The destination is a topic for which there is a durable subscriber.

Note: Designer populates **Destination Type** automatically if you selected a destination from the list of existing destinations on the JMS provider.

5. In the **JMS Message Selector** column, click . In the Enter JMS Message Selector dialog box, enter the expression that you want to use to receive a subset of messages from this destination and click **OK**.

   For more information about creating a JMS message selector, see "Creating a Message Selector" on page 640.

6. If you specified the destination type as **Topic (Durable Subscriber)**, in the **Durable Subscriber Name** column, do one of the following:

   - Enter a name for the durable subscriber.
   - If the JMS connection alias creates a connection on webMethods Broker, Universal Messaging, or Nirvana click to select from a list of existing durable subscribers for the topic. In the Durable Subscriber List dialog box select the durable subscriber and click **OK**.

     If the durable subscriber that you want this JMS trigger to use does not exist, you can create it by entering in the name in the **Durable Subscriber Name** column. The name must be unique for the connection where the connection name is the client ID of the JMS connection alias. webMethods Broker, Universal Messaging, or Nirvana will create the durable subscriber name using the client ID of the JMS connection alias and the specified durable subscriber name.

   Note: Designer populates **Durable Subscriber Name** automatically if you selected a Topic (Durable Subscriber) destination from the list of existing destinations on webMethods Broker or Universal Messaging.

7. If you want the JMS trigger to ignore messages sent using the same JMS connection alias as the JMS trigger, select the check box in the **Ignore Locally Published** column. This property applies only when the Destination Type is **Topic** or **Topic (Durable Subscriber)**.

   Note:
   If the JMS connection alias specified for this trigger has the **Create New Connection per Trigger** option enabled, then **Ignore Locally Published** will not work. For the JMS trigger to ignore locally published messages, the publisher and subscriber must share the same connection. When the JMS connection alias uses multiple connections per trigger, the publisher and subscriber will not share the same connection.

8. Repeat this procedure for each destination from which you want the JMS trigger to receive messages.

9. Click **File > Save**.

Notes:

- If you specify a new durable subscriber name and the JMS connection alias that the JMS trigger uses to retrieve messages is configured to manage destinations, Integration Server creates a durable subscriber for the topic when the JMS trigger is first enabled.
- If you specify a destination type of **Topic (Durable Subscriber)** but do not specify a durable subscriber name, Designer changes the destination type to **Topic** when you save the JMS trigger.

Creating a Destination on the JMS Provider

If the JMS connection alias that the JMS trigger uses to retrieve messages is configured to manage destinations, you can create a destination on the JMS provider while using the JMS trigger editor.

Keep the following points in mind when creating destinations using Designer:

- The JMS connection alias used by the JMS trigger must use Universal Messaging, Nirvana, or webMethods Broker as the JMS provider.

  Note:
  Prior to version 9.5 SP1, Software AG Universal Messaging was named webMethods Nirvana.

- The JMS connection alias used by the JMS trigger must be configured to manage destinations.
- The JMS connection alias must be enabled when you work with the JMS trigger.
- If the JMS connection alias creates a connection on a webMethods Broker in a webMethods Broker cluster, you will not be able to create a destination at the webMethods Broker.

To create a destination on the JMS provider

1. In the Package Navigator view of Designer, open the JMS trigger that uses a JMS connection alias that connects to the JMS provider on which you want to create the destinations.
2. In the Trigger Settings tab, under **JMS destinations and message selectors**, click .
3. In the **Destination Name** column, click .
4. In the Destination List dialog box, click **Create New Destination**.
5. In the Create New Destination dialog box, provide the following information:

In this field...	Specify...
Destination Name	A name for the destination.
Destination Key	A name for the destination key. If you do not specify a destination key, Integration Server uses the destination name as the destination key. In the Destination List, when a destination has a destination key Designer displays the destination name using this format: *destinationKey* (*destinationName*)
Destination Type	The type of destination. Select one of the following:

Select...	To...
Queue	The destination is a queue.

In this field...	Specify...	
		This is the default.
	Topic	The destination is a topic.
	Topic (Durable Subscriber)	The destination is a topic for which you want to create a durable subscriber.
Durable Subscriber Name	A name for the durable subscriber. The name must be unique for the connection, where the connection name is the client ID of the JMS connection alias. The JMS provider (webMethods Broker, Universal Messaging, or Nirvana) will create the durable subscriber name using the client ID of the JMS connection alias and the specified durable subscriber name. This field only applies if the destination is **Topic (Durable Subscriber)**.	
Order By	How webMethods Broker distributes messages received by this destination. This field only apples if the JMS provider used y the trigger JMS connection alias is the webMethods Broker and the destination is **Queue**.	
	Select...	**To...**
	Publisher	Distribute messages received by this destination one at a time in the order in which they were received from the publisher.
	None	Distribute the messages received by this destination in any order. This is the default.
	Note: An order mode of publisher maps to a serial message processing mode. An order mode of none maps to a concurrent message processing mode.	

6. Click **OK** to create the destination.

7. If you want the current JMS trigger to retrieve messages from the new destination, select the destination and click **OK**.

   Designer adds the destination to the **JMS destinations and message selectors** list. If the **Order By** mode for the new destination does not match the existing message processing mode, Designer prompts you to change the processing mode.

Notes:

- Integration Server adds the new destination to the webMethods Broker as a shared-state client.
- If you specify a destination type of **Topic (Durable Subscriber)** but do not specify a durable subscriber name, Designer changes the destination type to **Topic** when you save the JMS trigger.

About Durable and Non-Durable Subscribers

When a JMS trigger receives messages from a topic, you can specify whether or not the JMS trigger is a durable subscriber.

A *durable subscriber* establishes a durable subscription with a unique identity on the JMS provider. A *durable subscription* allows subscribers to receive all the messages published on a topic, including those published while the subscriber is inactive (for example, if the JMS trigger is disabled). When the associated JMS trigger is disabled, the JMS provider holds the messages in guaranteed storage. If a durable subscription already exists for the specified durable subscriber on the JMS provider, this service resumes the subscription.

A *non-durable subscription* allows subscribers to receive messages on their chosen topic only if the messages are published while the subscriber is active. A non-durable subscription lasts the lifetime of its message consumer. Note that non-durable subscribers cannot receive messages in a load-balanced fashion.

Creating a Message Selector

If you want the JMS trigger to receive a subset of messages from a specified destination, create a message selector. A message selector is an expression that specifies the criteria for the messages in which the JMS trigger is interested.

The JMS provider applies the message selector to messages it receives. If the selector evaluates to true, the message is sent to the JMS trigger. If the selector evaluates to false, the message is not sent to the JMS trigger.

By creating message selectors, you can delegate some filtering work to the JMS provider. This can preserve Integration Server resources that otherwise would have been spent receiving and processing unwanted messages.

The message selector must use the message selector syntax specified in the Java Message Service standard. The message selector can reference header and property fields in the JMS message only.

Note:
If you want to filter on the contents of the JMS message body, write a local filter. Integration Server evaluates a local filter after the JMS trigger receives the message from the JMS provider. Only standard JMS triggers can use local filters.

Adding Routing Rules to a Standard JMS Trigger

The routing rule specifies the service that Integration Server invokes when the standard JMS trigger receive a message from a destination.

To add a routing rule to a standard JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger.
2. In the Trigger Settings tab, under **Message routing,** click to add a new routing rule.
3. In the **Name** column, type a name for the routing rule. By default Designer assigns the first rule the name "Rule 1".
4. In the **Service** column, click to navigate to and select the service that you want to invoke when Integration Server receives messages from the specified destinations.
5. In the **Local Filter** column, click to enter the filter that you want Integration Server to apply to messages this JMS trigger receives. For more information about creating a local filter, see "Creating a Local Filter" on page 641.
6. Click **File > Save**.

Creating a Local Filter

You can further refine the messages received and processed by a standard JMS trigger by creating local filters. A local filter specifies criteria for the contents of the message body. Integration Server applies a local filter to a message after the JMS trigger receives the message from the JMS provider. If the message meets the filter criteria, Integration Server executes the trigger service specified in the routing rule. If the message does not meet the filter criteria, Integration Server discards the message and acknowledges the message to the JMS provider.

If a JMS trigger contains multiple routing rules to support ordered service execution, you can use local filters to process a series of messages in a particular order. For more information about ordered service execution, see "Building Standard JMS Triggers with Multiple Routing Rules" on page 643.

When creating a local filter, you can omit the *JMSMessage* document from the filter expression even though it is part of the pipeline provided to the JMS trigger service. For example, a filter that matches those messages where the value of the *myField* field is "XYZ" would look like the following:

```
%properties/myField% == "XYZ"
```

Note that even though the properties field is a child of the *JMSMessage* document, the *JMSMessage* document does not need to appear in the filter expression.

The following filter matches those messages where the *data* document within the *JMSMessage/body* document contains a field named *myField* whose value is "A":

```
%body/data/myField% == "A"
```

Note:
When receiving a batch of messages, Integration Server evaluates the local filter against the first message in the batch only. Integration Server does not apply the filter to subsequent messages in the batch. For more information about batch processing, see "About Batch Processing for Standard JMS Triggers" on page 651.

Managing Destinations and Durable Subscribers on the JMS Provider through Designer

When editing a JMS trigger in Designer, you can create and manage destinations and durable subscribers on Software AG Universal Messaging, webMethods Nirvana, or webMethods Broker. Specifically, you can do the following:

- Create a destination.
- Create and delete a durable subscriber.
- Select the destination from which you want the JMS trigger to receive messages from a list of existing destinations.
- Select a durable subscriber that you want the JMS trigger to use from a list of existing durable subscribers for a specified topic.
- Change the Shared State or Order By mode for a queue or durable subscriber by changing the message processing mode of the JMS trigger. You can do this only when webMethods Broker is the JMS provider only.

Designer uses the JMS connection alias specified by the JMS trigger to make the changes on the JMS provider. To manage destinations on the JMS provider, the JMS connection alias that the JMS trigger uses must be

- Configured to manage destinations
- Enabled when you create and edit the JMS trigger.
- To manage destinations on webMethods Broker, Integration Server must be version 8.0 SP1 or higher.
- To manage destinations on Universal Messaging, Integration Server must be version 9.0 SP1 or higher.

  Note:
  Prior to version 9.5 SP1, Software AG Universal Messaging was named webMethods Nirvana.

For a complete list of the requirements for using Designer to manage destinations and durable subscribers on the JMS provider, see *webMethods Integration Server Administrator's Guide*.

Note:
The ability to use Designer to manage JMS destinations on webMethods Broker, Nirvana, and Universal Messaging is a design-time feature. In a production environment, this functionality should be disabled.

Modifying Destinations or Durable Subscribers via a JMS Trigger in Designer

If a JMS trigger uses a JMS connection alias that is configured to manage destinations, you can modify the destination or durable subscribers while editing a JMS trigger. Changes to destinations or durable subscriptions can result in unused durable subscriptions on the JMS provider. Changing destinations can make the JMS trigger out of sync with the destination. For example, when using the webMethods Broker, modifying the destination could result in out of sync Shared State or Order By mode settings.

When you make a change that results in a change to a destination or durable subscriber, Designer informs you about the necessary change and then prompts you to confirm making change to the destination or durable subscriber on the JMS provider.

For example, if you change the name of the durable subscriber for a Topic (Durable Subscriber) destination, Designer displays a message stating, "By making this change the trigger will no longer subscribe to durable subscriber *oldDurableSubscriberName*. Would you like to remove this durable subscriber from the JMS provider?" If you confirm the change, Integration Server removes the durable subscriber from webMethods Broker. If you do not confirm the change, the durable subscriber will remain on webMethods Broker. You will need to use the webMethods Broker interface in My webMethods to remove the durable subscriber.

Note:
If another client, such as another JMS trigger, currently connects to the queue or durable subscriber that you want to modify or remove, then Integration Server cannot update or remove the queue or durable subscriber. If the JMS provider is webMethods Broker, updates must be made through My webMethods. If the JMS provider is Universal Messaging, updates must be made through Universal Messaging Enterprise Manager. If the JMS provider is Nirvana, updates must be made through Nirvana Enterprise Manager.

For more information about managing destinations and durable subscriptions on the JMS provider, see "Managing Destinations and Durable Subscribers on the JMS Provider through Designer " on page 642.

Building Standard JMS Triggers with Multiple Routing Rules

A JMS trigger can contain more than one routing rule. Each routing rule can specify a different local filter and a different service to invoke.

You might create multiple routing rules so that a JMS trigger processes a group of messages in a specific order. Each routing rule might execute a different trigger service based on the contents or type of message received. When a JMS trigger receives a message, Integration Server determines which service to invoke by evaluating the local filters for each routing rule.

Integration Server evaluates the routing rules in the same order in which the rules appear in the editor. It is possible that a message could satisfy more than one routing rule. However, Integration Server executes only the service associated with the first satisfied routing rule and ignores the remaining routing rules. Therefore, the order in which you list routing rules on the editor is important.

You might want to use multiple routing rules to control service execution when a service that processes a message depends on successful execution of another service. For example, to process a purchase order, you might create one service that adds a new customer record to a database, another that adds a customer order, and a third that bills the customer. The service that adds a customer order can only execute successfully if the new customer record has been added to the database. Likewise, the service that bills the customer can only execute successfully if the order has been added. You can ensure that the services execute in the necessary order by creating a trigger that contains one routing rule for each expected message.

Note:
SOAP-JMS triggers do not have routing rules.

Guidelines for Building a JMS Trigger that Performs Ordered Service Execution

Use the following general guidelines to build a JMS trigger that performs ordered service execution.

- Because the JMS provider cannot guarantee message order across destinations, the JMS trigger must specify a single destination. That is, the JMS trigger cannot include a join.
- Each routing rule, except the last one, must contain a local filter. For example, you might create a filter based on a custom property that the sending client adds to the message. Integration Server uses the local filters to differentiate between the messages. Without a local filter, only the first routing rule would ever execute.
- Routing rules must appear in the order in which you want the messages to be processed. Each routing rule must have a unique name.
- Set the **Processing mode** property to serial to ensure that the Integration Server processes the messages in the same order in which the JMS trigger receives them. Serial processing ensures that the services that process the messages do not execute at the same time.
- Set **Max batch messages** to 1 (the default). When a trigger service processes a batch of messages, Integration Server only applies the filter to the first message in the batch.

Important:
Messages must be sent to JMS provider in the same order in which you want the messages to be processed.

Enabling or Disabling a JMS Trigger

You can enable or disable a JMS trigger.

Note:
If you disable a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Integration Server will not retrieve any messages for those web service descriptors.

To enable or disable a JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger that you want to enable or disable.

2. In the Properties view, under **General**, set the **Enabled** property to one of the following:

Select...	To...
True	Enable a JMS trigger that is currently disabled.
False	Disable a JMS trigger that is currently enabled.

3. Click **File > Save**.

Notes:

- When you disable a JMS trigger, Integration Server interrupts any server threads that are processing messages. If the JMS trigger is currently processing messages, Integration Server waits 3 seconds before forcing the JMS trigger to stop processing messages. If it does not complete within 3 seconds, Integration Server stops the message consumer used to receive messages for the JMS trigger and closes the JMS consumer. At this point the server thread for the JMS trigger may continue to run to completion. However, the JMS trigger will not be able to acknowledge the message when processing completes. If the message is persistent, this can lead to duplicate messages.
- You can disable one or more JMS triggers using the pub.triggers:disableJMSTriggers service.
- You can enable one or more JMS triggers using the pub.triggers:enableJMSTriggers service.
- You can enable, disable, and suspend one or more JMS triggers using Integration Server Administrator.

JMS Trigger States

A JMS trigger can have one of the following states:

Trigger State	Description
Enabled	The JMS trigger is available. A JMS trigger must be enabled for it to receive and process messages. An enabled trigger can have a status of “Not Running” which means that it would not receive and process messages. Reasons that an enabled JMS trigger

Trigger State	Description
	can be disabled include: a disabled JMS connection alias, an exception thrown by the trigger, and trigger failure at startup. JMS trigger status can be seen on the **Settings > Messaging > JMS Trigger Management** page in Integration Server Administrator.
Disabled	The JMS trigger is not available. Integration Server neither retrieves nor processes messages for the JMS trigger. The JMS trigger remains in this state until you enable the trigger.
Suspended	The JMS trigger is running and connected to the JMS provider. Integration Server has stopped message retrieval, but continues processing any messages it has already retrieved. Integration Server enables the JMS trigger automatically upon server restart or when the package containing the JMS trigger reloads. **Note:** You can suspend a JMS trigger using Integration Server Administrator or the pub.triggers:suspendJMSTriggers service.

Setting an Acknowledgement Mode

Acknowledgment mode indicates how Integration Server acknowledges messages received on behalf of a JMS trigger. A message is not considered to be successfully consumed until it is acknowledged.

Note:
The **Acknowledgement mode** property is not available for transacted JMS triggers. That is, if the JMS connection alias is of type XA_TRANSACTION or LOCAL_TRANSACTION, Designer does not display the **Acknowledgement mode** property.

To set an acknowledgment mode

1. In the Package Navigator view of Designer, open the JMS trigger for which you want to set the acknowledgment mode.

2. In the Properties view, under **General**, select one of the following for **Acknowledgement mode**:

Select...	To...
CLIENT_ACKNOWLEDGE	Acknowledge or recover the message only after the JMS trigger processes the message completely. This is the default.

Select...	To...
DUPS_OK_ACKNOWLEDGE	Lazily acknowledge the delivery of messages. This may result in the delivery of duplicate messages.
AUTO_ACKNOWLEDGE	Automatically acknowledge the message when it is received by the JMS trigger. Integration Server will acknowledge the message before the trigger completes processing. The JMS provider cannot redeliver the message if Integration Server becomes unavailable before message processing completes.

3. Click **File > Save**.

About Join Time-Outs

When you create a standard JMS trigger that receives messages from two or more destinations), you create a join. Consequently, you need to specify a join time-out. A *join time-out* specifies how long Integration Server waits for additional messages to fulfill the join. Integration Server starts the join time-out period when it receives the first message that satisfies the join.

The implications of a join time-out are different depending on the join type.

Note:
You need to specify a join time-out only when the join type is **All (AND)** or **Only one (XOR)**. You do not need to specify a join time-out for an **Any (OR)** join.

Join Time-Outs for All (AND) Joins

A join time-out for an **All (AND)** join specifies how long Integration Server waits for messages from all of the destinations specified in the join.

When a JMS trigger receives a message that satisfies part of an All (AND) join, Integration Server stores the message. Integration Server waits for the JMS trigger to receive messages from the remaining destinations specified in the join. Only messages with the same activation ID as the first received message will satisfy the join.

If Integration Server receives messages from all of the destinations specified in the join before the time-out period elapses, Integration Server executes the service specified in the routing rule. If Integration Server doe not receive messages from all of the destinations before the time-out period elapses, Integration Server discards the messages and writes a log entry.

When the time-out period elapses, the next message that satisfies the **All (AND)** join causes the time-out period to start again.

Join Time-Outs for Only One (XOR) Joins

A join time-out for an **Only one (XOR)** join specifies how long Integration Server discards instances of the other messages received from the specified destinations.

When a JMS trigger receives a message that satisfies part of an **Only one (XOR)** join, Integration Server executes the service specified in the routing rule. Integration Server starts the join time-out when the JMS trigger receives the message. For the duration of the time-out period, Integration Server discards any messages the JMS trigger receives from a destination specified in the JMS trigger. Integration Server only discards those messages with the same activation ID as the first message.

When the time-out period elapses, the next message that the JMS trigger receives that satisfies the **Only one (XOR)** join causes the trigger service to execute and the time-out period to start again.

Setting a Join Time-Out

When configuring JMS trigger properties, you can specify whether a join times out and if it does, what the time-out period should be. The time-out period indicates how long Integration Server waits for messages from the other destinations specified in the join after Integration Server receives the first message.

Note:
You need to specify a join time-out only when the join type is **All (AND)** or **Only one (XOR)**. You do not need to specify a join time-out for an **Any (OR)** join.

To set a join time-out

1. In the Package Navigator view of Designer, open the JMS trigger for which you want to set the join time-out.

2. In the Properties view, under **General**, next to **Join expires**, select one of the following:

Select...	To...
True	Specify that Integration Server should stop waiting for messages from other destinations in the join after the time-out period elapses. In the **Expire after** property, specify the length of the join time-out period. The default join time-out period is 1 day.
False	Specify that the join does not expire. Integration Server should wait indefinitely for messages from the additional destinations specified in the join condition. Set the **Join expires** property to **False** only if you are confident that all of the messages will be received eventually. **Important:**

Select...	To...
	A join is persisted across server restarts.

3. Click **File > Save**.

About Execution Users for JMS Triggers

For a JMS trigger, the execution user indicates which credentials Integration Server should use when invoking services associated with the JMS trigger. When a client invokes a service via an HTTP request, Integration Server checks the credentials and user group membership of the client against the Execute ACL assigned to the service. Integration Server performs this check to make sure that the client is allowed to invoke that service. When a JMS trigger executes, however, Integration Server invokes the service when it receives a message rather than as a result of a client request. Integration Server does not associate user credentials with a message. You can specify which credentials Integration Server should supply when invoking a JMS trigger service by setting an execution user for a JMS trigger.

You can instruct Integration Server to invoke a service using the credentials of one of the predefined user accounts (Administrator, Default, Developer, Replicator). You can also specify a user account that you or another server administrator defined. When Integration Server receives a message for the JMS trigger, Integration Server uses the credentials for the specified user account to invoke the service specified in the routing rule.

Assigning an Execution User to a JMS Trigger

Make sure that the user account you select includes the credentials required by the execute ACL assigned to the services associated with the JMS triggers.

To assign an execution user for a JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger for which you want to assign the execution user.

2. In the Properties view, under **General**, in the **Execution user** property, type the name of the user account whose credentials Integration Server uses to execute a service associated with the JMS trigger. You can specify a locally defined user account or a user account defined in a central or external directory.

3. Click **File > Save**.

About Message Processing

Message processing determines how Integration Server processes the messages received by the JMS trigger. You can specify serial processing or concurrent processing.

- In serial processing, Integration Server processes messages received by a JMS trigger one after the other in the order in which the messages were received from the JMS provider.
- In concurrent processing, Integration Server processes messages received from the JMS provider in parallel.

Serial Processing

In serial processing, Integration Server processes messages received by a JMS trigger one after the other in the order in which the messages were received from the JMS provider. Integration Server uses a single thread for receiving and processing a message for a serial JMS trigger. Integration Server evaluates the first message it receives, determines which routing rule the message satisfies, and executes the service specified in the routing rule. Integration Server waits for the service to finish executing before processing the next message received from the JMS provider.

If you want to process messages in the same order in which JMS clients sent the messages to the JMS provider, you will need to configure the JMS provider to ensure that messages are received by the JMS trigger in the same order in which the messages are published.

For information about using serial JMS triggers in a cluster to process messages from a single destination in publishing order, see the *Using webMethods Integration Server to Build a Client for JMS*.

Tip:
If your trigger contains multiple routing rules to handle a group of messages that must be processed in a specific order, use serial processing.

Concurrent Processing

In concurrent processing, Integration Server processes messages received from the JMS provider in parallel. That is, Integration Server processes as many messages for the JMS triggers as it can at the same time, using a separate server thread to process each message. Integration Server does not wait for the service specified in the routing rule to finish executing before it begins processing the next message. You can specify the maximum number of messages Integration Server can process concurrently. This equates to specifying the maximum number of server threads that can process messages for the JMS trigger at one time.

Concurrent processing provides faster performance than serial processing. Integration Server processes the received messages more quickly because it can process more than one message for the trigger at a time. However, the more messages Integration Server processes concurrently, the more server threads it dispatches, and the more memory the message processing consumes.

Additionally, for JMS triggers with concurrent processing, Integration Server does not guarantee that messages are processed in the order in which they are received.

A concurrent trigger can connect to the JMS provider through multiple connections, which can increase trigger throughout. For more information about multiple connections, refer to "Using Multiple Connections to Retrieve Messages for a Concurrent JMS Trigger" on page 653.

Message Processing and Message Consumers

Integration Server uses a consumer to receive messages for a JMS trigger. This consumer encapsulates the actual javax.jms.MessageConsumer and javax.jms.Session. The type of message processing affects how Integration Server uses consumers to receive messages.

Serial JMS triggers have one consumer and will use one thread from the server thread pool to receive and process a message.

Concurrent JMS triggers use a pool of consumers to receive and process messages. Each consumer uses one thread from the server thread pool to receive and process a message. For a concurrent JMS trigger, the **Max execution threads** property specifies how many threads can be used to process messages for the trigger at one time. For concurrent JMS triggers, Integration Server also dedicates a thread to managing the pool of consumers. Consequently, the maximum number of threads that can be used by a JMS trigger is equal to the **Max execution threads** value plus 1. For example, a concurrent JMS trigger configured to use 10 threads at a time can use a maximum of 11 server threads.

When there are multiple connections to the webMethods Broker, the threads are divided among the connections. Therefore, if a trigger is configured so that **Connection count** is 2 and **Max execution threads** is set to 10, each connection will have 5 threads plus 1, for a total of 12 threads.

Message Processing and Load Balancing

Load balancing allows multiple consumers on one or more Integration Servers to retrieve and process messages concurrently. Load balancing is necessary for concurrent JMS triggers regardless of whether or not they are running in a cluster of Integration Servers. This is because concurrent JMS triggers use multiple consumers. Each consumer receives a message from the JMS provider, processes the message, and acknowledges the message to the JMS provider. Each consumer needs to consume a message from the same destination, but not process any duplicate message. For information about configuring load-balancing, see *webMethods Integration Server Administrator's Guide*.

About Batch Processing for Standard JMS Triggers

You can configure a standard JMS trigger and its associated trigger service to process a group or "batch" of messages at one time. Batch processing can be an effective way of handling a high volume of small messages for the purposes of persisting them or delivering them to another back-end resource. For example, you might want to take a batch of messages, create a packet of SAP IDocs, and send the packet to SAP with a single call. Alternatively, you might want to insert multiple messages into a database at one time using only one insert. The trigger service processes the messages as a unit as opposed to in a series.

The **Max batch messages** property indicates the maximum number of messages that the trigger service can receive at one time. For example, if the **Max batch messages** property is set to 5, Integration Server passes the trigger service up to 5 messages received by the JMS trigger to process during a single execution.

Integration Server uses one consumer to receive and process a batch of messages. During pre-processing, Integration Server checks the maximum delivery count for each message and, if exactly-once processing is configured, determines whether or not the message is a duplicate. Integration Server then bundles the message into a single IData and passes it to the trigger service. If the message has exceeded the maximum delivery count or is a duplicate message, Integration Server does not include it in the message batch sent to the trigger service.

Note:
The watt.server.jms.trigger.maxDeliveryCount property determines the maximum number of times the JMS provider can deliver a message to a JMS trigger.

Integration Server acknowledges all the messages received in a batch from the JMS provider at one time. This includes messages that failed pre-processing. As described by the Java Message Service standard, when a client acknowledges one message, the client acknowledges all of the messages received by the session. Because Integration Server uses a consumer that includes a javax.jms.MessageConsumer and a javax.jms.Session, when Integration Server acknowledges one message in the batch, it effectively acknowledges all the messages received in the batch.

If a batch of messages is not acknowledged or they are recovered back to the JMS provider, the JMS provider can redeliver all of the messages in the batch to the JMS trigger. However, when using webMethods Broker, Integration Server can acknowledge individual messages that fail pre-processing.

Guidelines for Configuring Batch Processing

When configuring JMS trigger for batch processing, keep the following in mind:

- The trigger service must be coded to handle multiple messages as input. That is, the trigger service must use the pub.jms.batchTriggerSpec as the service signature.
- When receiving a batch of messages, Integration Server evaluates the local filter in the routing rule against the first message in the batch only.
- A transacted JMS trigger can be used for batch processing if the JMS connection alias used by the trigger connects to a JMS provider that supports reuse of transacted JMS sessions. If the JMS provider does not support reuse of transacted JMS sessions, set **Max batch processing** to 1.

  Consult the documentation for your JMS provider to determine whether or not the JMS provider supports the reuse of transacted JMS sessions. Note that webMethods Broker version 8.2 and higher, Software AG Universal Messaging version 9.5 SP1 and higher, and webMethods Nirvana version 7 and higher support the reuse of transacted JMS sessions.
- A JMS trigger that contains an All (AND) or Only one (XOR) join cannot use batch processing.
- SOAP-JMS triggers cannot process messages in batches.

Using Multiple Connections to Retrieve Messages for a Concurrent JMS Trigger

You can configure a concurrent JMS trigger to obtain multiple connections to the JMS provider. Multiple connections can improve trigger throughput. Keep in mind, however, that each connection used by the JMS trigger requires a dedicated Integration Server thread, regardless of the current throughput.

For a JMS trigger to have multiple connections to the JMS provider, the JMS connection alias used by the trigger must be configured to create a new connection for each trigger. For more information about JMS connection aliases, refer to *webMethods Integration Server Administrator's Guide*.

A concurrent JMS trigger can use multiple connections to retrieve messages from a JMS provider. For a trigger to use multiple connections, the following must be true:

- The JMS trigger must be configured for concurrent processing. Serial JMS triggers cannot use multiple connections.
- The JMS trigger must receive messages from a queue or from a topic with a durable subscriber. JMS triggers that receive messages from non-durable subscribers (topics) cannot use multiple connections.
- The JMS trigger must not have the **Ignore locally published** option selected when the JMS connection alias is configured to use the **Create New Connection per Trigger** option. For the JMS trigger to ignore locally published messages, the publisher and subscriber must share the same connection. When the JMS connection alias uses multiple connections per trigger, the publisher and subscriber will not share the same connection.
- The JMS connection alias used by the JMS trigger must be configured to create an individual connection for each trigger. To configure a JMS alias to create individual connections for each JMS trigger, select the **Create New Connection per Trigger** option on the **Settings > Messaging > JMS Settings > JMS Connection Alias** screen on Integration Server Administrator.

Note:
When using multiple connections to the webMethods Broker, Integration Server uses a different client ID for each JMS trigger that uses the JMS connection alias. However, when Integration Server connects to other JMS providers, it uses the same client ID for each connection. Some JMS providers do not permit multiple connections to use the same client ID to retrieve messages from a Topic with a durable subscriber. Review the JMS provider documentation before configuring the use of multiple connections for a JMS connection alias and any concurrent JMS triggers that use the JMS connection alias.

Retrieving Multiple Messages for a JMS Trigger with Each Request

You can instruct Integration Server to retrieve multiple messages for a JMS trigger with each request by using the prefetch cache. When a JMS trigger is configured to use the prefetch cache,

Integration Server retrieves multiple messages for the trigger each time Integration Server requests more messages from the webMethods Broker. When the JMS trigger needs a new message to process, the JMS trigger retrieves the message from the local, prefetched cache instead of requesting a new message from the webMethods Broker. Use of the prefetch cache may improve performance of the JMS trigger because it reduces the time spent retrieving messages for the JMS trigger.

Using the prefetch cache is most likely to improve performance for JMS triggers that process many small messages and have trigger services that execute quickly. If the JMS trigger receives large messages or the JMS trigger has long-running trigger services, using the prefetch cache may increase the overall time needed to retrieve and process a message. For JMS triggers that fit this use case, including concurrent JMS triggers, reducing the number of prefetched messages may actually decrease the time needed to retrieve and process a message. You may need to set the number of prefetched messages to 1 (one).

Note:
This prefetch cache can be used with JMS triggers that receive messages from webMethods Broker only.

The use of the prefetch cache for a JMS trigger and the number of messages Integration Server might retrieve with each request are determined by the **Max prefetch size** property for the JMS trigger and the value of the watt.server.jms.trigger.maxPrefetchSize parameter.

- When the **Max prefetch size** property is greater than 0, Integration Server uses the prefetch cache with the JMS trigger. The **Max prefetch size** property value specifies the number of messages that Integration Server might retrieve and cache for the trigger. The default is 10.
- When the **Max prefetch size** property is set to -1, Integration Server uses the prefetch cache with the JMS trigger. The watt.server.jms.trigger.maxPrefetchSize parameter value determines how many messages Integration Server might retrieves and cache for the JMS trigger.
- When the **Max prefetch size** property is set to 0, Integration Server does not use the prefetch cache with the JMS trigger.

When the prefetch cache is in use and the number of messages retrieved by Integration Server is greater than one, the same server thread might process all of the messages retrieved by the prefetch request. This is true even for concurrent JMS triggers. The first thread for the concurrent JMS trigger processes the first set of prefetched messages. The second thread for the concurrent JMS trigger processes the second set of prefetched messages.

For example, suppose that the number of available messages is 22, **Max execution threads** is 4, and **Max prefetch size** is 10. In the initial request for messages, the first server thread may retrieve 10 messages. The same server thread will process these first 10 messages. The second server thread may retrieve 10 messages, all of which will be processed by the second server thread. The third server thread may retrieve the remaining 2 messages, both of which will be processed by the third server thread. While the concurrent JMS trigger can use up to 4 server threads, Integration Server might use only 3 server threads to retrieve and process messages due to the way in which a JMS trigger processes prefetched messages. A concurrent JMS trigger will use all of the configured execution threads to process messages only when the number of messages on the webMethods Broker is greater than the number of messages that can be prefetched.

Note:

When you are working with a cluster of Integration Servers, the prefetch behavior might appear at first to be misleading. For example, suppose that you have a cluster of two Integration Servers. Each Integration Server contains the same JMS trigger. Twenty messages are sent to a destination from which JMS trigger receives messages. It might be expected the JMS trigger on Integration Server 1 will receive the first message, the JMS trigger on Integration Server 2 will receive the second message, and so forth. However, what may happen is that the JMS trigger on Integration Server 1 will receive the first 10 messages and the JMS trigger on Integration Server 2 will receive the second 10 messages.

Configuring Message Processing

Keep the following points in mind when configuring message processing for a JMS trigger:

- You can configure a standard JMS trigger and its associated trigger service to process a group or "batch" of messages at one time. For information about batch processing, see "About Batch Processing for Standard JMS Triggers" on page 651 and "Guidelines for Configuring Batch Processing" on page 652.
- If the JMS provider from which the JMS trigger retrieves messages does not support concurrent access by durable subscribers, you must set the **Max execution threads** property to 1 for the concurrent JMS trigger. Consult the documentation for your JMS provider for more information.
- Non-durable subscribers, i.e., JMS triggers that subscribe to topics but do not specify a durable subscriber, cannot receive messages in a load-balanced fashion. Because it is possible for a JMS trigger using a non-durable subscriber to process duplicates of a message, set **Max execution threads** to 1.
- For a destination that acts as a shared state client, the serial processing mode corresponds to a shared state order mode of publisher; a concurrent processing mode corresponds to a shared state order mode of none.
- If you use webMethods Broker as the JMS provider, changing the message processing mode for a JMS trigger can create a mismatch with the corresponding destination on the webMethods Broker. If you do not use Designer to make the changes, you need to use the webMethods Broker interface of My webMethods to update the destination.
- A concurrent JMS trigger can use multiple connections to retrieve messages from the JMS provider. For information about requirements for using multiple connections, see "Using Multiple Connections to Retrieve Messages for a Concurrent JMS Trigger" on page 653.
- You can only use the **Max prefetch** property with webMethods Broker.

To configure message processing for a JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger for which you want to specify message processing.

2. In the Properties view, under **Messaging processing**, next to **Processing mode**, select one of the following:

Select...	To...
Serial	Specify that Integration Server should process messages received by the trigger one after the other.
Concurrent	Specify that Integration Server should process multiple messages for this trigger at one time. In the **Max execution threads** property, specify the maximum number of messages that Integration Server can process concurrently.

3. If you want this trigger to perform batch processing, next to **Max batch messages**, specify the maximum number of messages that the trigger service can receive at one time. If you do not want the trigger to perform batch processing, leave this property set to 1. The default is 1.

4. If you want this trigger to use multiple connections to receive messages from the JMS provider, next to **Connection count**, specify the number of connections you want the JMS trigger to make to the JMS provider. The default is 1.

5. If you want Integration Server to use the prefetch cache with this JMS trigger, in the Properties view, under **webMethods Broker** do one of the following for **Max prefetch size**:

   - Specify the number of messages you want Integration Server to retrieve and cache for this JMS trigger. The default is 10 messages.
   - Specify -1 if you want the value of watt.server.jms.trigger.maxPrefetchSize parameter to determine how many messages Integration Server retrieves and caches for the JMS trigger.
   - Specify 0 if you do not want to use the prefetch cache with this JMS trigger.

6. Click **File > Save**.

   If the destination is Queue or Topic (Durable Subscriber) and the JMS trigger is connected to the queue or durable subscriber, Designer prompts you to update the corresponding destination on the webMethods Broker with the changed shared state order mode, click **Yes** to update the destination. Click **No** to skip the destination update. Note that messages might be lost while Designer and Integration Server make the update because Integration Server deletes and recreates the subscription as part of the update.

   Note:
   A JMS trigger is connected to the webMethods Broker when the specified JMS connection alias is enabled and connected to the webMethods Broker.

Fatal Error Handling for Non-Transacted JMS Triggers

You can specify that Integration Server suspend a JMS trigger automatically if a fatal error occurs during trigger service execution. A fatal error occurs when the trigger service ends because of an exception.

If a trigger service ends because of an exception, and you configured the JMS trigger to suspend on fatal errors, Integration Server suspends the trigger and acknowledges the message to the JMS provider. The JMS trigger remains suspended until one of the following occurs:

- You enable the trigger using the pub.trigger:enableJMSTriggers service.
- You enable the trigger using Integration Server Administrator.
- Integration Server restarts or the package containing the trigger reloads. (When Integration Server suspends a trigger because of a fatal error, Integration Server considers the change to be temporary. For more information about temporary vs. permanent state changes for triggers, see *webMethods Integration Server Administrator's Guide*.)

Automatic suspension of a trigger can be especially useful for serial triggers that are designed to process a group of messages in a particular order. If the trigger service ends in error while processing the first message, you might not want the trigger to proceed with processing the subsequent messages in the group. If Integration Server automatically suspends the trigger, you have an opportunity to determine why the trigger service did not execute successfully.

Important:
If you disable or suspend a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Integration Server will not retrieve any messages for those web service descriptors until the trigger is enabled.

You can handle the exception that causes the fatal error by configuring Integration Server to generate JMS retrieval failure events for fatal errors and by creating an event handler that subscribes to JMS retrieval failure events. Integration Server passes the event handler the contents of the JMS message as well as information about the exception.

Integration Server handles fatal errors for transacted JMS differently than for non-transacted JMS triggers. For information about fatal error handling for transacted JMS triggers, see "Fatal Error Handling for Transacted JMS Triggers" on page 671.

Configuring Fatal Error Handling for Non-Transacted JMS Triggers

To configure fatal error handling for a non-transacted JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger for which you want to specify document processing.

2. In the Properties view, under **Fatal error handling**, set the **Suspend on error** property to **True** if you want Integration Server to suspend the trigger when a trigger service ends with an error. Otherwise, select **False**. The default is **False**.

3. Click **File > Save**.

Transient Error Handling for Non-Transacted JMS Triggers

When building a JMS trigger, you can specify what action Integration Server takes when the trigger service fails because of a transient error caused by a run-time exception. A *transient error* is an error that arises from a temporary condition that might be resolved or corrected quickly, such as the unavailability of a resource due to network issues or failure to connect to a database. Because the condition that caused the trigger service to fail is temporary, the trigger service might execute successfully if Integration Server waits and then re-executes the service.

A *run-time exception* (specifically, an ISRuntimeException) occurs in the following situations:

- The trigger service catches and wraps a transient error and then re-throws it as an ISRuntimeException.
- The web service operation that processes the message received by a SOAP-JMS trigger catches and wraps a transient error and then re-throws it as an ISRuntimeException.

  Note:
  For a service handler invoked by a SOAP-JMS trigger, Integration Server treats all errors as fatal. Service handlers invoked by SOAP-JMS triggers cannot be retried.

- A pub.jms:send, pub.jms:sendAndWait, or pub.jms:reply service fails because a resource (such as the JNDI provider or JMS provider) is not available.
- If the JMS provider is not available, and the settings for the pub.jms* service indicate that Integration Server should write messages to the client side queue, Integration Server does not throw an ISRuntimeException.
- A transient error occurs on the back-end resource for an adapter service. Adapter services built on Integration Server 6.0 or later, and based on the ART framework, detect and propagate exceptions that signal a retry automatically if a transient error is detected on their back-end resource.

Note:
A web service connector that sends a JMS message can throw an ISRuntimeException, such as when the JMS provider is not available. However, Integration Server automatically places the ISRuntimeException in the *fault* document returned by the web service connector. If you want the parent flow service to catch the transient error and re-throw it as an ISRuntimeException, you must code the parent flow service to check the *fault* document for an ISRuntimeException and then throw an ISRuntimeException explicitly.

You can also configure Integration Server and/or a JMS trigger to handle transient errors that occur during trigger preprocessing. The trigger preprocessing phase encompasses the time from when a trigger first receives a message from it's local queue on Integration Server to the time the trigger service executes.

For more information about transient error handling for trigger preprocessing, see "Transient Error Handling During Trigger Preprocessing" on page 737.

About Retry Behavior for Trigger Services

When you configure transient error handling for a non-transacted JMS trigger, you specify the following retry behavior:

- Whether Integration Server should retry trigger services for the standard JMS trigger. Keep in mind that a trigger service can retry only if it is coded to throw ISRuntimeExceptions. For more information, see "Service Requirements for Retrying a Trigger Service" on page 659.
- For a SOAP-JMS trigger, whether Integration Server should retry web service operation that throw and an ISRuntimeException.

  Note: Integration Server does not apply the SOAP-JMS trigger transient error handling behavior to service handlers executed as part of processing web services. Integration Server treats all errors thrown by service handler as fatal errors.
- The maximum number of retry attempts Integration Server should make for each trigger service.
- The time interval between retry attempts.
- How to handle a retry failure. That is, you can specify what action Integration Server takes if all the retry attempts are made and the trigger service or web service operation still fails because of an ISRuntimeException. For more information about handling retry failures, see "Handling Retry Failure" on page 660.

Service Requirements for Retrying a Trigger Service

To be eligible for retry, the trigger service or web service operation must do one of the following to catch a transient error and re-throw it as an ISRuntimeException:

- If the trigger service or web service operation is a flow service, the trigger service must invoke pub.flow:throwExceptionForRetry. For more information about the pub.flow:throwExceptionForRetry, see the *webMethods Integration Server Built-In Services Reference*.
- If the trigger service or web service operation is written in Java, the service can use com.wm.app.b2b.server.ISRuntimeException(). For more information about constructing ISRuntimeExceptions in Java services, see the *webMethods Integration Server Java API Reference* for the com.wm.app.b2b.server.ISRuntimeException class.

When a service invokes a pub.jms* service that sends a JMS message and the service fails because a resource needed by the pub.jms* service is not available, Integration Server automatically detects and propagates an ISRuntimeException.

Adapter services built on Integration Server 6.0 or later, and based on the ART framework, detect and propagate exceptions that signal a retry if a transient error is detected on their back-end resource. This behavior allows for the automatic retry when the service functions as a trigger service.

Note: Integration Server does not retry a trigger service that fails because a ServiceException occurred. A ServiceException indicates that there is something functionally wrong with the service. A service can throw a ServiceException using the EXIT step.

Handling Retry Failure

Retry failure occurs for a standard JMS trigger when Integration Server makes the maximum number of retry attempts and the trigger service still fails because of an ISRuntimeException. Retry failure occurs for a SOAP-JMS trigger when Integration Server makes the maximum number of retry attempts to process a web service operation and the operation still fails because of an ISRuntimeException.

When you configure retry properties, you can specify one of the following actions to determine how Integration Server handles retry failure for a trigger.

- **Throw exception.** When Integration Server exhausts the maximum number of retry attempts, Integration Server treats the last trigger service or web service operation failure as a service error. This is the default behavior.
- **Suspend and retry later.** When Integration Server reaches the maximum number of retry attempts, Integration Server suspends the trigger and then retries the trigger service or web service operation at a later time.

Overview of Throw Exception for Retry Failure

The following table provides an overview of how Integration Server handles retry failure when the **Throw exception** option is selected.

Step	Description
1	Integration Server makes the final retry attempt and the trigger service or web service operation fails because of an ISRuntimeException.
2	Integration Server treats the last trigger service or web service operation failure as a ServiceException.
3	Integration Server rejects the message. If the message is persistent, Integration Server returns an acknowledgement to the JMS provider.
4	Integration Server generates a JMS retrieval failure event if the watt.server.jms.trigger.raiseEventOnRetryFailure property is set to true (the default).
5	If the JMS trigger is configured to suspend on error when a fatal error occurs, Integration Server suspends the JMS trigger. Otherwise, Integration Server processes the next message for the JMS trigger.

In summary, the default retry failure behavior (**Throw exception**) rejects the message and allows the trigger to continue with message processing when retry failure occurs for a trigger service.

Overview of Suspend and Retry Later for Retry Failure

The following table provides more information about how the **Suspend and retry later** option works.

Step	Description
1	Integration Server makes the final retry attempt and the trigger service or web service operation fails because of an ISRuntimeException.
2	Integration Server suspends the JMS trigger temporarily. **Note:** The change to the trigger state is temporary. Message processing will resume for the trigger if Integration Server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads. You can also enable triggers manually using Integration Server Administrator or by invoking the pub.trigger:enableJMSTriggers service. **Important:** If you disable or suspend a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Integration Server will not retrieve any messages for those web service descriptors until the SOAP-JMS trigger is enabled.
3	Integration Server recovers the message back to the JMS provider. This indicates that the required resources are not ready to process the message and makes the message available for processing at a later time. For serial triggers, it also ensures that the message maintains its position at the top of trigger queue.
4	Optionally, Integration Server schedules and executes a resource monitoring service. A *resource monitoring service* is a service that you create to determine whether the resources associated with a trigger service are available. A resource monitoring service returns a single output parameter named *isAvailable*.
5	If the resource monitoring service indicates that the resources are available (that is, the value of *isAvailable* is true), Integration Server enables the trigger. Message processing and message retrieval resume for the JMS trigger. If the resource monitoring service indicates that the resources are not available (that is, the value of *isAvailable* is false), Integration Server waits a short time interval (by default, 60 seconds) and then re-executes the resource monitoring service. Integration Server continues executing the resource monitoring service periodically until the service indicates the resources are available. **Tip:** You can change the frequency with which the resource monitoring service executes by modifying the value of the watt.server.jms.trigger.monitoringInterval property.

Step	Description
6	After Integration Server resumes the JMS trigger, Integration Server passes the message to the trigger. The trigger and trigger service (or web service operation) process the message just as they would any message received by the JMS trigger. **Note:** At this point, the retry count is set to 0 (zero).

In summary, the **Suspend and retry later** option provides a way to resubmit the message programmatically. It also prevents the trigger from retrieving and processing other messages until the cause of the transient error condition has been remedied.

Configuring Transient Error Handling for a Non-Transacted JMS Trigger

The transient error handling and retry behavior that you specify for a non-transacted JMS trigger determines how Integration Server handles retry failure caused by transient errors during trigger service execution. The selected behavior also determines how Integration Server handles transient errors that occur during trigger preprocessing.

For more information about transient error handling for trigger preprocessing, see "Transient Error Handling During Trigger Preprocessing" on page 737.

Note:
If you do not configure service retry for a trigger, set the **Max retry attempts** property to 0. Because managing service retries creates extra overhead, setting this property to 0 can improve the performance of services invoked by the trigger.

To configure transient error handling for a non-transacted JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger for which you want to configure retry behavior.

2. In the Properties view, under **Transient error handling**, in the **Max retry attempts field**, specify the maximum number of times Integration Server should attempt to re-execute the trigger service. The default is 0 retries (the trigger service does not retry).

3. In the **Retry interval** property, specify the time period the Integration Server waits between retry attempts. The default is 10 seconds.

4. Set the **On retry failure** property to one of the following:

Select...	To...
Throw exception	Specify that Integration Server should throw a service exception when the last allowed retry attempt ends because of an ISRuntimeException. This is the default.
Suspend and retry later	Specify that Integration Server should recover the message back to the JMS provider and suspend the trigger when the last allowed retry attempt ends because of an ISRuntimeException. **Note:** If you want Integration Server to automatically enable the trigger when the trigger's resources become available, you must provide a resource monitoring service that Integration Server can execute to determine when to resume the trigger.

5. If you selected **Suspend and retry later**, then in the **Resource monitoring service** property specify the service that Integration Server should execute to determine the availability of resources associated with the trigger service. Multiple triggers can use the same resource monitoring service. For information about building a resource monitoring service, see *Using webMethods Integration Server to Build a Client for JMS*.

6. Click **File > Save**.

Notes:

- Standard JMS triggers and services can both be configured to retry. When a trigger invokes a service (that is, the service functions as a trigger service), Integration Server uses the trigger retry properties instead of the service retry properties.
- SOAP-JMS triggers and services used as operations in provider web service descriptors can both be configured to retry. When a web service operation processes a message received by a SOAP-JMS trigger, Integration Server uses the trigger retry properties instead of the service (operation) retry properties.
- Integration Server does not retry service handlers invoked by a SOAP-JMS trigger.
- When Integration Server retries a trigger service and the trigger service is configured to generate audit data on error, Integration Server adds an entry to the audit log for each failed retry attempt. Each of these entries will have a status of "Retried" and an error message of "Null". However, if Integration Server makes the maximum retry attempts and the trigger service still fails, the final audit log entry for the service will have a status of "Failed" and will display the actual error message. Integration Server makes the audit log entry regardless of which retry failure option the trigger uses.
- Integration Server generates the following journal log message between retry attempts:

  [ISS.0014.0031D] Service *serviceName* failed with ISRuntimeException. Retry *x* of *y* will begin in *retryInterval* milliseconds.

- You can invoke the pub.flow:getRetryCount service within a trigger service to determine the current number of retry attempts made by Integration Server and the maximum number of retry attempts allowed for the trigger service. For more information about the pub.flow:getRetryCount service, see the *webMethods Integration Server Built-In Services Reference*.

Exactly-Once Processing for JMS Triggers

Within Integration Server, exactly-once processing is a facility that ensures one-time processing of a persistent message by a JMS trigger. The trigger does not process duplicates of the message. Integration Server provides exactly-once processing when all of the following are true:

- The message is persistent.
- The JMS trigger has an acknowledgement mode set to `CLIENT_ACKNOWLEDGE`.
- Exactly-once properties are configured for the JMS trigger.

Note: Software AG recommends that if you want to use exactly-once processing for JMS triggers subscribing to topics, make sure the topic uses a durable subscriber.

Duplicate Detection Methods for JMS Triggers

Integration Server ensures exactly-once processing by performing duplicate detection and by providing the ability to retry trigger services. *Duplicate detection* determines whether the current message is a copy of one previously processed by the trigger.

Duplicate messages can be introduced in to the webMethods system in the following situations:

- The sending client sends the same message more than once.
- When receiving persistent messages from the JMS provider, Integration Server and the JMS provider lose connectivity before the JMS trigger processes and acknowledges the message. The JMS trigger will receive the message again when the connection is restored.

Integration Server uses duplicate detection to determine the message's status. The message status can be one of the following:

- **New.** The message is new and has not been processed by the trigger.
- **Duplicate.** The message is a copy of one already processed the trigger.
- **In Doubt.** Integration Server cannot determine the status of the message. The trigger may or may not have processed the message before.

To resolve the message status, Integration Server evaluates, in order, one or more of the following:

- **Delivery count** indicates how many times the JMS provider has delivered the message to the JMS trigger.
- **Document history database** maintains a record of all persistent message IDs processed by JMS triggers that have an acknowledgment mode of `CLIENT_ACKNOWLEDGE` and for which exactly-once processing is configured.

- **Document resolver service** is a service created by a user to determine the message status. The document resolver service can be used instead of or in addition to the document history database.

The steps that Integration Server takes to determine a message's status depend on the exactly-once properties configured for the JMS trigger.

Note:
For detailed information about exactly-once processing for messages received by JMS triggers, see *Using webMethods Integration Server to Build a Client for JMS*.

Configuring Exactly-Once Processing for a JMS Trigger

Configure exactly-once processing for a JMS trigger when you want the trigger to process persistent messages once and only once. If it is acceptable for a trigger service to process duplicates of a message, you should not configure exactly-once processing for the trigger.

Keep the following points in mind when configuring exactly-once processing:

- Integration Server can perform exactly-once processing for persistent messages only. The sending client must set the *JMSDeliveryMode* to persistent.
- The JMS trigger must specify `CLIENT_ACKNOWLEDGE` for the acknowledgement mode.
- You do not need to configure all three methods of duplicate detection. However, if you want to ensure exactly-once processing, you must use a document history database or implement a custom solution using the document resolver service.

  A document history database offers a simpler approach than building a custom solution and will typically catch all duplicate messages. There may be exceptions depending on your implementation. For more information about these exceptions, see "Building a Transacted JMS Trigger" on page 668. To minimize these exceptions, it is recommended that you use a history database and a document resolver service.
- Stand-alone Integration Servers cannot share a document history database. Only a cluster of Integration Servers or a non-clustered group of Integration Servers can (and must) share a document history database.
- Make sure the duplicate detection window set by the **History time to live** property is long enough to catch duplicate messages but does not cause the document history database to consume too many server resources. If sending JMS clients reliably send messages once, you might use a smaller duplicate detection window. If the JMS clients are prone to sending duplicate messages, consider setting a longer duplicate detection window.
- If you intend to use a document history database as part of duplicate detection, you must first install the document history database component and associate it with a JDBC connection pool. For instructions, see *Installing Software AG Products*.

Note:
For detailed information about exactly-once processing for messages received by JMS triggers, see *Using webMethods Integration Server to Build a Client for JMS*.

To configure exactly-once processing for a JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger for which you want to configure exactly-once processing.

2. In the Properties view, under **Exactly Once**, set the **Detect duplicates** property to **True**.

3. To use a document history database as part of duplicate detection, do the following:

   a. Set the **Use history** property to **True**.

   b. In the **History time to live** property, specify how long the document history database maintains an entry for a message processed by this trigger. This value determines the length of the duplicate detection window.

4. To use a service that you create to resolve the status of In Doubt messages, specify that service in the **Document resolver service** property.

5. Click **File > Save**.

Disabling Exactly-Once Processing for a JMS Trigger

If you later determine that exactly-once processing is not necessary for a JMS trigger, you can disable it.

To disable exactly-once processing for a JMS trigger

1. In the Package Navigator view of Designer, open the trigger for which you want to configure exactly-once processing.

2. In the Properties view, under **Exactly Once**, set the **Detect duplicates** property to **False**.

   Designer disables the remaining exactly-once properties.

3. Click **File > Save**.

Debugging a JMS Trigger

To debug and test a JMS trigger you can:

- Instruct Integration Server to produce an extra level of verbose logging. You can enable debug trace logging for all JMS triggers or for individual JMS triggers
- Send messages to which the JMS trigger subscribes to the JMS provider. You can create a service that sends the messages. Alternatively, you can create a launch configuration that

publishes a JMS message that contains an instance of a specified IS document type to the JMS provider.

Enabling Trace Logging for All JMS Triggers

You can configure Integration Server to generate additional debug and trace logging for all JMS triggers.

Note:
For the increased logging to appear in the server log, you must set the logging level for server facility 0134 JMS Subsystem to Trace.

To enable debug trace logging for all JMS triggers

1. Open Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the Navigation panel, click **Extended**.
3. Click **Edit Extended Settings**.
4. Under **Extended Settings**, type the following:

```
watt.server.jms.debugTrace=true
```

5. Click **Save Changes**.
6. Suspend and then enable all JMS triggers.

   For information about suspending and enabling all JMS triggers at one time, see *webMethods Integration Server Administrator's Guide*.

Enabling Trace Logging for a Specific JMS Trigger

You can configure Integration Server to generate additional debug and trace logging for a specific JMS triggers.

Note:
For the increased logging to appear in the server log, you must set the logging level for server facility 0134 JMS Subsystem to Trace.

To enable debug trace logging for a specific JMS trigger

1. Open Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the Navigation panel, click **Extended**.

3. Click **Edit Extended Settings**.

4. Under **Extended Settings**, type the following:

   ```
   watt.server.jms.debugTrace.triggerName=true
   ```

   Where *triggerName* is the fully qualified name of the trigger in the format *folder.subfolder:triggerName*.

5. Click **Save Changes**.

6. Disable and then enable the trigger.

Building a Transacted JMS Trigger

A *transacted JMS trigger* is a JMS trigger that executes within a transaction. A *transaction* is a logical unit of work composed of one or more interactions with one or more resources. The interactions within a transaction are either all committed or all rolled back. A transaction either entirely succeeds or has no effect at all.

For a transacted JMS trigger, Integration Server uses a transacted JMS connection alias to receive messages from the JMS provider and to process the messages. A JMS connection alias is considered to be transacted when it has a transaction type of XA TRANSACTION or LOCAL TRANSACTION.

The execution of a transacted JMS trigger is an implicit transaction. In an implicit transaction, Integration Server starts and completes the transaction automatically, without the need for executing any of the transaction management services.

Integration Server starts the implicit transaction when it uses the specified transacted JMS connection alias to connect to the JMS provider and receive messages for the transacted JMS trigger. Integration Server implicitly commits or rolls back the transaction based on the success or failure of the trigger service.

- Integration Server commits the transaction if the trigger service executes successfully.
- Integration Server rolls back the transaction if the trigger service fails with an ISRuntimeException (a transient error). For detailed information about how Integration Server handles a transient error within a transaction, see "Transient Error Handling for Transacted JMS Triggers" on page 673.
- Integration Server rolls back the transaction if the trigger service fails with a Service Exception (a fatal error). For detailed information about how Integration Server handles a fatal error within a transaction, see "Fatal Error Handling for Transacted JMS Triggers" on page 671.

Because Integration Server handles the transaction implicitly, you do not need to use any of the transaction management services, such as pub.art.transaction:startTransaction, in the trigger service. However, if the trigger service includes a nested transaction, you can use the transaction management services to explicitly manage the nested transaction.

Like a non-transacted JMS trigger, a transacted JMS trigger specifies a destination from which it would like to receive documents and specifies routing rules to process messages it receives.

However, a transacted JMS trigger has some prerequisites as well as some properties that are different from a non-transacted JMS trigger.

Prerequisites for a Transacted JMS Trigger

Before you build a transacted JMS trigger, make sure the following points are true:

- A transacted JMS connection alias exists. A JMS connection alias is considered to be transacted when it has a transaction type of XA TRANSACTION or LOCAL TRANSACTION.

  Note:
  A transacted JMS connection alias cannot be assigned to a JMS trigger if a cluster policy is applied to the connection factory used by the JMS connection alias.

- The WmART package is installed and enabled.

Properties for Transacted JMS Triggers

Integration Server and Designer provide different properties for a transacted JMS trigger than for a non-transacted JMS trigger. The following list identifies properties that are specific to transacted JMS triggers, specific to non-transacted JMS triggers, or apply to both but must be set to a particular value for transacted JMS triggers.

- For transacted JMS triggers, message acknowledgement is handled by the transaction; the acknowledgement mode does not apply. Consequently, Designer does not display the **Acknowledgement mode** property for a transacted JMS trigger.
- A transacted JMS trigger can only use Any (OR) joins, for which you do not need to specify a join time-out. Because All (AND) and Only one (XOR) joins cannot be used, Designer does not display the **Join expires** and **Expire after** properties for a transacted JMS trigger.
- A transacted JMS trigger can be used for batch processing if the JMS connection alias used by the trigger connects to a JMS provider that supports reuse of transacted JMS sessions. If the JMS provider does not support reuse of transacted JMS sessions, set **Max batch processing** to 1.

  Consult the documentation for your JMS provider to determine whether or not the JMS provider supports the reuse of transacted JMS sessions. Note that webMethods Broker version 8.2 and higher, Software AG Universal Messaging version 9.5 SP1 and higher, and webMethods Nirvana version 7 and higher support the reuse of transacted JMS sessions.
- Because a transaction is an all or nothing situation, a trigger service cannot retry a message if a trigger service ends because of a transient error. Designer does not display the retry properties (**Max retry attempts**, **Retry interval**, and **On retry failure**) for a transacted JMS trigger.
- You can specify how Integration Server handles a transient error that causes the transaction to be rolled back. Designer displays an **On transaction rollback** property that you can use to specify whether Integration Server simply recovers the message from the JMS provider or whether it suspends the JMS trigger in addition to recovering the message. For more information

about transient error handling for transacted JMS triggers, see "Transient Error Handling for Transacted JMS Triggers" on page 673.

Steps for Building a Transacted JMS Trigger

Building a transacted JMS trigger is a process that involves the following basic stages.

Stage 1 Create a new JMS trigger on Integration Server.

Stage 2 Specify a JMS connection alias with a transaction type of XA TRANSACTION or LOCAL TRANSACTION.

Stage 3 Specify the destination (queues or topics) on the JMS provider from which you want to receive messages. You also specify any message selectors that you want the JMS provider to use to filter messages for the JMS trigger.

If this a SOAP-JMS trigger, you can specify one destination only.

Stage 4 For a standard JMS trigger, create routing rules and specify the services that Integration Server invokes when the JMS trigger receives messages.

SOAP-JMS triggers do not use routing rules.

Stage 5 Set the following JMS trigger properties:

Property name...	Description
Enabled	Enables or disables a JMS trigger as follows: ■ If set to **True**, enables a JMS trigger that is currently disabled. ■ If set to **False**, disables a JMS trigger that is currently enabled.
Execution user	Name of the user account whose credentials Integration Server uses to execute a service associated with the JMS trigger.
Message processing	Specifies whether Integration Server should process messages serially or concurrently. When set to: ■ **Serial**, Integration Server processes messages received by the trigger one after the other. ■ **Concurrent**, Integration Server processes multiple messages for the trigger at one time.
Fatal error handling > Suspend on error	Specifies whether you want Integration Server to suspend the trigger when a trigger service ends with an error. Select **True** or **False**.

Transient error handling Specifies how Integration Server responds when a transaction is rolled back due to a transient error that occurs during processing of a transacted JMS trigger. When the **On transaction rollback** property is set to:

- **Recover only**, Integration Server recovers the message after a transaction is rolled back due to a transient error. This is the default.
- **Suspend and recover**, Integration Server suspends the JMS trigger and recovers the message after a resource monitoring service indicates that the resources needed by the trigger service are available.

Exactly once Specifies whether you want the trigger to process persistent messages once and only once. Set **Detect duplicates** to **True** to configure exactly once processing.

Permissions In Designer, select the ACLs that you want to assign for each level of access as follows:

- For the **List ACL** permission, specify the ACL whose allowed member can see that the element exists and view the element's metadata (input, output, etc.).
- For the **Read ACL**, specify the ACL whose allowed member can view the source code and metadata of the element.
- For the **Write ACL**, specify the ACL whose allowed member can lock, check out, edit, rename, and delete the element.
- For the **Execute ACL**, specify the ACL whose allowed member can execute the service. This level of access only applies to services and web service descriptors.

Stage 6 Test and debug the JMS trigger. For more information, see "Debugging a JMS Trigger" on page 666.

Fatal Error Handling for Transacted JMS Triggers

You can specify that Integration Server suspend a transacted JMS trigger automatically if a fatal error occurs during trigger service execution. For a standard JMS trigger, a fatal error occurs when the trigger service ends because of a ServiceException. For a SOAP-JMS trigger, a fatal error occurs when the web service operation ends because of a ServiceException.

When a transacted JMS trigger is configured to suspend when a fatal error occurs, Integration Server does the following when the trigger service or web service operation ends with a ServiceException:

Step	Description
1	The trigger service for a transacted JMS trigger fails because of a ServiceException. Or, a web service operation invoked via a transacted SOAP-JMS trigger fails because of a ServiceException.
2	Integration Server rolls back the entire transaction and Integration Server recovers the message back to the JMS provider. The JMS provider marks the message as redelivered and increments the value of the *JMSXDeliveryCount* property in the JMS message.
3	If the JMS trigger is configured to use a document history database for exactly-once processing, Integration Server adds an entry with a status of "completed" for the message to the document history database. Because Integration Server does not acknowledge the message when it is rolled back, the JMS provider makes the message available for redelivery to the JMS trigger. However, a message that causes a trigger service to end because of a Service Exception typically does not process successfully upon redelivery. Integration Server adds the "completed" entry so that the message is treated as a duplicate when it is received from the JMS provider. The message is rejected after it is resent. If the JMS trigger does not use a document history database, Integration Server continues to receive and attempt message processing until the message processes successfully or the maximum delivery count has been met. The maximum delivery count determines the maximum number of time the JMS provider can deliver the message to the JMS trigger. It is controlled by the watt.server.jms.trigger.maxDeliveryCount property.
4	Integration Server suspends the JMS trigger. **Important:** If you disable or suspend a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Integration Server will not retrieve any messages for those web service descriptors until the trigger is enabled.
5	The JMS trigger remains suspended until one of the following occurs: ■ You enable the trigger using the pub.trigger:enableJMSTriggers service. ■ You enable the trigger using Integration Server Administrator. ■ Integration Server restarts or the package containing the trigger reloads. (When Integration Server suspends a trigger because of a fatal error, Integration Server considers the change to be temporary. For more information about temporary vs. permanent state changes for triggers, see *webMethods Integration Server Administrator's Guide*.)

You can handle the exception that causes the fatal error by configuring Integration Server to generate JMS retrieval failure events for fatal errors and by creating an event handler that subscribes to JMS retrieval failure events. Integration Server passes the contents of the JMS message and exception information to the event handler.

Configuring Fatal Error Handling for Transacted JMS Triggers

To configure fatal error handling for a transacted JMS trigger

1. In the Package Navigator view of Designer, open the JMS trigger for which you want to specify document processing.

2. In the Properties view, under **Fatal error handling**, set the **Suspend on error** property to **True** if you want Integration Server to suspend the trigger when a trigger service ends with an error. Otherwise, select **False**. The default is **False**.

3. Configure exactly-once processing for the JMS trigger. For more information about configuring exactly-once processing, see "Configuring Exactly-Once Processing for a JMS Trigger" on page 665.

4. Click **File > Save**.

Transient Error Handling for Transacted JMS Triggers

When building a transacted JMS trigger, you can specify what action Integration Server takes when a transient error causes a trigger service or a web service operation to fail and the entire transaction is rolled back.

A *transient error* is an error that arises from a temporary condition that might be resolved or corrected quickly, such as the unavailability of a resource due to network issues or failure to connect to a database. A transient error is caused by a run-time exception. A *run-time exception* (specifically, an ISRuntimeException) occurs in the following situations.

- The trigger service catches and wraps a transient error and then re-throws it as an ISRuntimeException.
- The web service operation that processes the message received by a SOAP-JMS trigger catches and wraps a transient error and then re-throws it as an ISRuntimeException.

  Note:
  For a service handler invoked by a SOAP-JMS trigger, Integration Server treats all errors as fatal. Service handlers invoked by SOAP-JMS triggers cannot be retried.

- The pub.jms:send, pub.jms:sendAndWait, or pub.jms:reply service fails because a resource (such as the JNDI provider or JMS provider) is not available.

  If the JMS provider is not available, and the settings for the pub.jms* service indicate that Integration Server should write messages to the client side queue, Integration Server does not throw an ISRuntimeException.

- A transient error occurs on the back-end resource for an adapter service. Adapter services built on Integration Server 6.0 or later, and based on the ART framework, detect and propagate

exceptions that signal a retry automatically if a transient error is detected on their back-end resource.

Note:
A web service connector that sends a JMS message can throw an ISRUntimeException, such as when the JMS provider is not available. However, Integration Server automatically places the ISRuntimeException in the *fault* document returned by the web service connector. If you want the parent flow service to catch the transient error and re-throw it as an ISRuntimeException, you must code the parent flow service to check the *fault* document for an ISRuntimeException and then throw an ISRuntimeException explicitly.

You can specify one of the following transient error handling options for a transacted JMS trigger:

- **Recover only.** After a transaction is rolled back, Integration Server receives the message from the JMS provider almost immediately. This is the default.
- **Suspend and recover.** After a transaction is rolled back, Integration Server suspends the JMS trigger and receives the message from the JMS provider at a later time.

You can also configure Integration Server and/or a JMS trigger to handle transient errors that occur during trigger preprocessing. The trigger preprocessing phase encompasses the time from when a trigger first receives a message from it's local queue on Integration Server b to the time the trigger service executes.

For more information about transient error handling for trigger preprocessing, see "Transient Error Handling During Trigger Preprocessing" on page 737.

Overview of Recover Only for Transaction Rollback

The following table provides an overview of how Integration Server handles transaction rollback when the **Recover Only** option is selected for a transacted JMS trigger.

Step	Description
1	The trigger service web service operation fails because of an ISRuntimeException.
2	Integration Server rolls back the entire transaction. When the transaction is rolled back, Integration Server recovers the message back to the JMS provider automatically. The JMS provider marks the message as redelivered and increments the delivery count (*JMSXDeliveryCount* field in the JMS message). At this point, a JMS provider typically makes the message available for immediate redelivery.
3	Integration Server receives the same message from the JMS provider and processes the message. Because Integration Server receives the message almost immediately after transaction roll back, it is likely that the temporary condition that caused the ISRuntimeException has not resolved and the trigger service will end with a transient error again.

Step	Description
	Consequently, setting **On transaction rollback** to **Recover only** could result in wasted processing. **Note:**Integration Server enforces a maximum delivery count, which determines the maximum number of time the JMS provider can deliver the message to the JMS trigger. If the maximum delivery count has been met, the JMS provider will not deliver the message to the JMS trigger. Instead, the JMS provider will acknowledge and remove the message. The maximum delivery count is controlled by the watt.server.jms.trigger.maxDeliveryCount property.

Overview of Suspend and Recover for Transaction Rollback

The following table provides an overview of how Integration Server handles transaction rollback when the **Suspend and recover** option is selected for a transacted JMS trigger.

Step	Description
1	The trigger service or web service operation fails because of an ISRuntimeException.
2	Integration Server rolls back the entire transaction. When the transaction is rolled back, Integration Server recovers the message back to the JMS provider automatically. The JMS provider marks the message as redelivered and increments the delivery count (*JMSXDeliveryCount* field in the JMS message).
3	Integration Server suspends the JMS trigger temporarily. The JMS trigger is suspended on this Integration Server only. If the Integration Server is part of a cluster, other servers in the cluster can retrieve and process messages for the trigger. **Important:** If you disable or suspend a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Integration Server will not retrieve any messages for those web service descriptors until the SOAP-JMS trigger is enabled. **Note:** The change to the trigger state is temporary. Message processing will resume for the trigger if Integration Server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads. You can also enable triggers manually using Integration Server Administrator or by invoking the pub.trigger:enableJMSTriggers service.
4	Optionally, Integration Server schedules and executes a resource monitoring service. A *resource monitoring service* is a service that you create to determine whether the resources associated with a trigger service are available. A resource monitoring service returns a single output parameter named *isAvailable*.

Step	Description
5	If the resource monitoring service indicates that the resources are available (that is, the value of *isAvailable* is true), Integration Server enables the trigger. Message processing and message retrieval resume for the JMS trigger. If the resource monitoring service indicates that the resources are not available (that is, the value of *isAvailable* is false), Integration Server waits a short time interval (by default, 60 seconds) and then re-executes the resource monitoring service. Integration Server continues executing the resource monitoring service periodically until the service indicates the resources are available. **Tip:** You can change the frequency at which the resource monitoring service executes by modifying the value of the watt.server.jms.trigger.monitoringInterval property.
6	After Integration Server resumes the JMS trigger, Integration Server receives the message from the JMS provider and processes the message. **Note:** If the maximum delivery count has been met, the JMS provider will not deliver the message to the JMS trigger. The maximum delivery count determines the maximum number of time the JMS provider can deliver the message to the JMS trigger. It is controlled by the watt.server.jms.trigger.maxDeliveryCount property.

Configuring Transient Error Handling for Transacted JMS Triggers

The transient error handling and transaction rollback behavior that you specify for a transacted JMS trigger determines how Integration Server handles transaction rollback caused by transient errors during trigger service execution. The selected behavior also determines how Integration Server handles transient errors that occur during trigger preprocessing.

For more information about transient error handling for trigger preprocessing, see "Transient Error Handling During Trigger Preprocessing" on page 737.

Use the following procedure to configure how Integration Server responds when a transaction is rolled back due to a transient error that occurs during processing of a transacted JMS trigger.

To configure transient error handling for a transacted JMS trigger

1. In the Package Navigator view of Designer, open the trigger for which you want to configure transient error handling.

2. In the Properties view, under **Transient error handling**, in the **On transaction rollback** property, select one of the following:

Select...	To...
Recover only	Specify that Integration Server recovers the message after a transaction is rolled back due to a transient error. This is the default.
Suspend and recover	Specify that Integration Server does the following after a transaction is rolled back due to a transient error: ■ Suspends the JMS trigger ■ Recovers the message after a resource monitoring service indicates that the resources needed by the trigger service are available.

3. If you selected **Suspend and recover**, in the **Resource monitoring service** property, specify the service that Integration Server should execute to determine the availability of resources associated with the trigger service or web service operation. Multiple triggers can use the same resource monitoring service.

4. Click **File > Save**.

33 Working with webMethods Messaging Triggers

- Overview of Building a webMethods Messaging Trigger 680
- Creating a webMethods Messaging Trigger 682
- Disabling and Enabling a webMethods Messaging Trigger 695
- About Join Time-Outs 696
- About Priority Message Processing 698
- About Execution Users for webMethods Messaging Triggers 700
- About Capacity and Refill Level for the webMethods Messaging Trigger Queue 701
- About Document Acknowledgements for a webMethods Messaging Trigger 704
- About Message Processing 705
- Fatal Error Handling for a webMethods Messaging Trigger 714
- About Transient Error Handling for a webMethods Messaging Trigger 716
- Exactly-Once Processing for webMethods Messaging Triggers 723
- Modifying a webMethods Messaging Trigger 726
- Deleting webMethods Messaging Triggers 727
- Running a webMethods Messaging Trigger with a Launch Configuration 729
- Debugging a webMethods Messaging Trigger 734

A webMethods messaging trigger subscribes to one or more publishable document types and processes instances of those document types. A webMethods messaging trigger can receive documents published to a webMethods messaging provider (Software AG Universal Messaging or webMethods Broker) or documents published locally by the Integration Server on which the trigger resides.

Each webMethods messaging trigger is composed of two basic pieces:

- A subscription to one or more publishable document types
- A service that processes instances of those publishable document types.

When a webMethods messaging trigger receives a document to which it subscribes from the messaging provider, Integration Server passes the document to the specified service and then invokes the service.

Note:
Prior to Integration Server and Software AG Designer versions 9.5 SP1, a webMethods messaging trigger was called a webMethods Broker/local trigger.

Note:
Information about webMethods messaging triggers is located in *webMethods Service Development Help* and *Publish-Subscribe Developer's Guide*. Both documents include the "Working with webMethods Messaging Triggers" on page 679 topic. *Publish-Subscribe Developer's Guide* contains information such as how webMethods messaging triggers work, how Integration Server receives documents from webMethods messaging triggers, how webMethods messaging triggers with join conditions work, and how Integration Server performs exactly-once processing.

Overview of Building a webMethods Messaging Trigger

Building a webMethods messaging trigger is a process that involves the following basic stages:

Stage 1 Create a new webMethods messaging trigger on Integration Server.

During this stage, you create the new webMethods messaging trigger on the Integration Server where you will do your development and testing. For more information, see "Creating a webMethods Messaging Trigger " on page 682.

Stage 2 Create one or more conditions for the webMethods messaging trigger.

During this stage, you create a trigger condition which associates a subscription to a publishable document types with a service that processes instances of that document types. You can also create filters to apply to incoming documents and select join types.

Stage 3 Set webMethods messaging trigger properties.

During this stage, you set parameters that configure the run-time environment of this webMethods messaging trigger, such as trigger queue capacity, document processing mode, fatal and transient error handling, exactly-once processing, and priority level.

Stage 4 Run the webMethods messaging trigger.

During this stage you can use the tools provided by Designer to run and debug the webMethods messaging trigger. For more information, see "Running a webMethods Messaging Trigger with a Launch Configuration" on page 729.

webMethods Messaging Trigger Requirements

AwebMethods messaging trigger must meet the following requirements:

- The webMethods messaging trigger contains at least one condition.
- Each condition in the webMethods messaging trigger specifies a unique name.
- Each condition in the webMethods messaging trigger specifies a service.
- Each condition in the webMethods messaging trigger specifies at least one publishable document type.
- If more than one condition in the webMethods messaging trigger specifies the same publishable document type and the trigger receives messages from the webMethods Broker, the filters in the conditions must be the same. Specifically, the contents of the **Filter** column must be identical for each condition that subscribes to the publishable document type. Software AG does not recommend using the same publishable document type in more than one condition in the same trigger when receiving messages from webMethods Broker.

  Note:
  Provider filters must be identical if multiple conditions in the same trigger specify the same publishable document type.

- If more than one condition in the webMethods messaging trigger specifies the same publishable document type and the trigger receives messages from Universal Messaging, the provider filters must be identical in each condition but the local filters can be different. Specifically, the contents of the **Provider Filter (UM)** column must be identical for each condition that subscribes to the publishable document type. The contents of the **Filter** column can be different.
- The webMethods messaging trigger contains no more than one join condition.
- The webMethods messaging trigger subscribes to publishable document types that use the same messaging connection alias. For the publishable document types to which the trigger subscribes, the value of the **Connection alias name** property can be:
  - The name of a specific messaging connection alias.
  - DEFAULT where the default messaging connection alias is the same as the alias specified for the other publishable document types to which the trigger subscribes.
- The syntax of a filter applied to a publishable document type is correct. That is, the filter in the **Filter** column must be valid. Integration Server does not validate the provider filter in the **Provider Filter** column.

When you save a webMethods messaging trigger, Integration Server evaluates the webMethods messaging trigger to make sure the webMethods messaging trigger is valid. If Integration Server determines that the webMethods messaging trigger or a condition in the webMethods messaging trigger is not valid, Designer displays an error message and prompts you to cancel the save or continue the save with a disabled webMethods messaging trigger.

Trigger Service Requirements

The service that processes a document received by a webMethods messaging trigger is called a *trigger service*. A condition specifies a single trigger service.

A trigger service for a webMethods messaging trigger must meet the following requirements:

- Before you can enable a webMethods messaging trigger, the trigger service must already exist on the same Integration Server.
- The input signature for the trigger service needs to have a document reference to the publishable document type.
- The name for this document reference must be the fully qualified name of the publishable document type. The fully qualified name of a publishable document type conforms to the following format: folder.subfolder:PublishableDocumentTypeName

  For example, suppose that you want a webMethods messaging trigger to associate the Customers:customerInfo publishable document type with the Customers:addToCustomerStore service. On the **Input/Output** tab of the service, the input signature must contain a document reference named Customers:customerInfo.
- If you intend to use the service in a join condition (a condition that associates multiple publishable document types with a service), the service's input signature must have a document reference for each publishable document type. The names of these document reference fields must be the fully qualified names of the publishable document type they reference.

Note:
An XSLT service cannot be used as a trigger service.

Creating a webMethods Messaging Trigger

When you create a webMethods messaging trigger, keep the following points in mind:

- The publishable document types and services that you want to use in conditions must already existwebMethods messaging trigger.
- A webMethods messaging trigger can subscribe to publishable document types only. A webMethods messaging trigger cannot subscribe to ordinary IS document types.
- A webMethods messaging trigger must meet the requirements specified in " webMethods Messaging Trigger Requirements" on page 681.

Important:

When you create webMethods messaging triggers, work on a stand-alone Integration Server instead of an Integration Server in a cluster or non-clustered group. Creating, modifying, disabling, and enabling webMethods messaging triggers on an Integration Server in a cluster or non-clustered group can create inconsistencies in the object that corresponds to the trigger on the messaging provider.

To create a webMethods messaging trigger

1. In the Package Navigator view of Designer, click **File > New > webMethods Messaging Trigger**.

2. In the Create a New webMethods Messaging Trigger dialog box, select the folder in which you want to save the webMethods messaging trigger.

3. In the **Element Name** field, type a name for the webMethods messaging trigger using any combination of letters, and/or the underscore character.

4. Click **Finish**.

   Designer generates the new webMethods messaging trigger and displays it in the Designer window.

5. Under **Condition detail**, build a condition to specify the document types to which the webMethods messaging trigger subscribes and the trigger services that execute when instances of those document types are received. For more information about creating conditions, see "Creating Conditions" on page 683.

6. In the Properties view, set properties for the webMethods messaging trigger.

7. Click **File > Save**.

Notes:

- Integration Server validates the webMethods messaging trigger before saving it. If Integration Server determines that the webMethods messaging trigger is invalid, Designer prompts you to save the webMethods messaging trigger in a disabled state. For more information about valid webMethods messaging trigger, see " webMethods Messaging Trigger Requirements" on page 681.
- You can also use the pub.trigger:createTrigger service to create a webMethods messaging trigger. For more information about this service, see the *webMethods Integration Server Built-In Services Reference*.

Creating Conditions

A condition associates one or more publishable document types with a single service. A webMethods messaging trigger subscribes to the publishable document type in a subscription.

The service, called a trigger services, processes instance of the document type received by the trigger.

A condition can be a simple condition or a join condition. A simple a condition associates one publishable document type with a service. A join associates more than one publishable document types with a service and specifies how the trigger handles the documents as a unit.

A webMethods messaging trigger must have at least one condition.

Keep the following points in mind when you create a condition for a webMethods messaging trigger:

- The publishable document types and services that you want to use in a condition must already exist.
- A webMethods messaging trigger can subscribe to publishable document types only. A webMethods messaging trigger cannot subscribe to ordinary IS document types.
- An XSLT service cannot be used as a trigger service.
- Conditions must meet additional requirements identified in " webMethods Messaging Trigger Requirements" on page 681.
- Trigger services must meet additional requirements identified in "Trigger Service Requirements" on page 682.
- If a webMethods messaging trigger subscribes to a publishable document type that is not in the same package as the trigger, create a package dependency on the package containing the publishable document type from the package containing the trigger. This ensures that Integration Server loads the package containing the publishable document type before loading the trigger.
- If a webMethods messaging trigger uses a trigger service that is not in the same package as the trigger, create a package dependency on the package containing the trigger service from the package containing the trigger. This ensures that Integration Server loads the package containing the service before loading the trigger.

To create a condition for a webMethods messaging trigger

1. In the Package Navigator view of the Service Development perspective, open the webMethods messaging trigger for which you want to set a condition.

2. Under **Conditions**, click to add a new condition.

3. Under **Condition detail**, in the **Name** field, type the name you want to assign to the condition. Designer automatically assigns each condition a default name such as `Condition1` or `Condition2`. You can keep this name or change it to a more descriptive one.

4. In the **Service** field, enter the fully qualified service name that you want to associate with the publishable document types in the condition. You can type in the service name, or click to navigate to and select the service.

5. Click under **Condition detail** to add a new document type subscription for this webMethods messaging trigger .

6. In the Select dialog box, select the publishable document types to which you want to subscribe. You can select more than one publishable document type by using the CTRL or SHIFT keys.

   Designer creates a row for each selected publishable document type. Designer enters the name of the messaging connection alias used by each publishable document type in the **Connection Alias** column.

7. In the **Filter** column next to each publishable document type, do the following:

   - If the publishable document type uses webMethods Broker as the messaging provider, specify a filter that you want Integration Server and/or webMethods Broker to apply to each instance of this publishable document type. For more information, see "Creating Filters for Use with webMethods Broker " on page 690.
   - If the publishable document type uses Universal Messaging as the messaging provider, specify the local filter that you want Integration Server to apply to each instance of the publishable document type received by the trigger. For more information, see "Creating Filters for Use with Universal Messaging " on page 687.

   Create the filter in the **Filter** column using the conditional expression syntax described in *webMethods Service Development Help*.

   Filters are optional for a trigger condition. For more information about filters, see "Using Filters with a Subscription" on page 687.

8. If the publishable document type uses Universal Messaging as the messaging provider, in the **Provider Filter (UM only)** column, enter the filter that you want Universal Messaging to apply to each instance of the publishable document type. Universal Messaging enqueues the document for the trigger only if the filter criteria is met. For information about the syntax for provider filters for Universal Messaging, see the Universal Messaging documentation. For more information about using filters in trigger conditions, see "Creating Filters for Use with Universal Messaging " on page 687.

9. If you specified more than one publishable document type in the condition, select a join type.

Join Type	Description
All (AND)	Integration Server invokes the trigger service when the server receives an instance of each specified publishable document type within the join time-out period. The instance documents must have the same activation ID. This is the default join type.
Any (OR)	Integration Server invokes the trigger service when it receives an instance of any one of the specified publishable document types.

Join Type	Description
Only one (XOR)	Integration Server invokes the trigger service when it receives an instance of any of the specified document types. For the duration of the join time-out period, Integration Server discards any instances of the specified publishable document types with the same activation ID.

10. Repeat this procedure for each condition that you want to add to the webMethods messaging trigger .

11. Click **File > Save**.

Notes:

- Integration Server validates the webMethods messaging trigger before saving it. If Integration Server determines that the webMethods messaging trigger is invalid, Designer prompts you to save the webMethods messaging trigger in a disabled state. For more information about valid webMethods messaging triggers, see " webMethods Messaging Trigger Requirements" on page 681.
- Integration Server establishes the subscription locally by creating a trigger queue for the webMethods messaging trigger .
- If the trigger subscribes to one or more publishable document types that use webMethods Broker as the messaging provider, one of the following happens upon saving the trigger.
  - If Integration Server is currently connected to the webMethods Broker, Integration Server registers the trigger subscription with the webMethods Broker by creating a client for the trigger on the webMethods Broker. Integration Server also creates a subscription for each publishable document type specified in the webMethods messaging trigger conditions and saves the subscriptions with the webMethods messaging trigger client. webMethods Broker validates the filters in the webMethods messaging trigger conditions when Integration Server creates the subscriptions.
  - If Integration Server is not currently connected to a webMethods Broker, the webMethods messaging trigger will only receive documents published locally. When Integration Server reconnects to a webMethods Broker, the next time Integration Server restarts Integration Server will create a client for the webMethods messaging trigger on the webMethods Broker and create subscriptions for the publishable document types identified in the webMethods messaging trigger conditions. webMethods Broker validates the filters in the webMethods messaging trigger conditions when Integration Server creates the subscriptions.
- If the trigger subscribes to a publishable document type that uses Universal Messaging as the messaging provider, one of the following happens upon saving the trigger.
  - If Integration Server is currently connected to Universal Messaging, Integration Server creates a named object on the channel that corresponds to the publishable document type.
  - If Integration Server is not currently connected to Universal Messaging, you need to synchronize the publishable document type with the provider when the connection becomes

available. Synchronizing creates the named object on the channel that corresponds to the publishable document type.

- If a publishable document type specified in a webMethods messaging trigger condition does not exist on the webMethods Broker (that is, there is no associated webMethods Broker document type), Integration Server still creates the trigger client on the webMethods Broker, but does not create any subscriptions. Integration Server creates the subscriptions when you synchronize (push) the publishable document type with the webMethods Broker.
- If a publishable document type specified in a webMethods messaging trigger condition does not exist on Universal Messaging, Designer displays an error stating that a channel does not exist for the specified document type.
- When creating a condition, you can specify the trigger service by dragging a service from Package Navigator view and dropping it in the **Service** field. Additionally, you can specify the document types to which the webMethods messaging trigger subscribes by dragging one or more document types from Package Navigator view and dropping them in the table in **Condition detail**.
- If you need to specify nested fields in the **Filter** field, you can copy a path to the **Filter** field from the document type. Select the field in the document type, right-click and select **Copy**. You can then paste into the **Filter** field. However, you must add % as a preface and suffix to the copied path.

Using Filters with a Subscription

You can further specify the documents that you want a trigger to receive and process by creating a filter for the condition. A filter specifies criteria that the published document must meet before the webMethods messaging trigger receives and/or processes the document. You can use the following types of filters:

- **Provider filter.** A provider filter is saved on the messaging provider. The messaging provider applies the filter when it receives the document from the publisher. If the document meets the filter criteria, the messaging provider enqueues the document for the subscribing trigger.
- **Local filter.** A local filter is saved on Integration Server. After a trigger receives a document, Integration Server applies the filter to the document. If the document meets the filter criteria, Integration Server executes the trigger.

How you create filters for a condition depends on the following:

- The messaging provider used by the publishable document type.
- If the messaging provider is Universal Messaging, the encoding type for the publishable document type.

Creating Filters for Use with Universal Messaging

If a webMethods messaging trigger subscribes to publishable document types associated with a Universal Messaging connection alias, you can create:

- A provider filter that Universal Messaging applies to the documents that it receives. Universal Messaging saves the filter along with the subscription to the document type. When Universal Messaging receives an instance of the publishable document type, Universal Messaging applies the filter. Universal Messaging enqueues the document for the trigger only if the filter criteria is met.

  Use the **Provider Filter (UM only)** column in the Condition detail table to specify a provider filter. For information about the syntax for provider filters for Universal Messaging, see the Universal Messaging documentation.

  When you save a trigger, Integration Server does not evaluate the syntax of the provider filter. Integration Server passes the filter directly to Universal Messaging.

  Note:
  If the trigger contains multiple conditions that subscribe to the same publishable document type, Integration Server does verify that the provider filters are identical upon save. If the supplied provider filters are identical, Integration Server saves the trigger. If the provider filters are not identical, Integration Server throws an exception and considers the trigger to be invalid.

- A local filter that Integration Server applies to the published document header or document body after the trigger receives the document. Use the **Filter** column in the Condition detail table to specify a local filter.

  Create the local filter using the conditional expression syntax described in *webMethods Service Development Help*.

  When you save a trigger, Integration Server evaluates the local filter to make sure it uses the proper syntax. If the syntax is correct, Integration Server saves the webMethods messaging trigger in an enabled state. If the syntax is incorrect, Integration Server saves the webMethods messaging trigger in a disabled state

Universal Messaging Provider Filters and Encoding Type

The encoding type specified for the publishable document type to which the webMethods messaging trigger subscribes determines the scope of the published document to which Universal Messaging applies the filter.

- When IData is the encoding type, Universal Messaging applies the filter to the custom header fields added to a published document via the *_properties* field. The provider filter allows the webMethods messaging trigger to indicate which documents it wants to receive based on the header contents.

- When protocol buffers is the encoding type, Universal Messaging applies the filter to the body of the document only.

  Because Integration Server includes the headers in the body of the published document as well as in the document header, you can still filter on the document headers when the encoding type is protocol buffers.

When using protocol buffers to encode messages, there are additional considerations for creating provider filters for Universal Messaging. Specifically, Universal Messaging *cannot* filter on:

- Fields that cannot be encoded as protocol buffers, including fields whose names contain characters that are not valid for protocol buffers, data types that are not supported by protocol buffers, and duplicate fields.

  For information about fields that cannot be represented as protocol buffers, see the information about setting an encoding type for a publishable document type in the *webMethods Service Development Help*.

- Undeclared fields that are not defined in the publishable document type but are in the published document.
- Fields that contain null values. Even if the field can be represented in protocol buffers, at run time, a null value cannot be included in a protocol buffer message. Consequently, Universal Messaging cannot apply a provider filter that checks for a null value.
- Any list data type where one of the elements in the list contains a null value.
- Fields whose values are references to another field.

While Universal Messaging cannot filter on the contents of the fields identified above, these fields and their contents will be passed through as an UnknownFieldSet which is represented as an IData byte array. A webMethods messaging trigger that receives the document will be able to decode the fields and include them in the pipeline.

For Universal Messaging to filter on protocol buffers, the following configuration properties must be set to true, the defaults, on Universal Messaging:

- **Global Values > ExtendedMessageSelector**
- **Protobuff Config > Filter ProtobufEvents**

Use Universal Messaging Enterprise Manager to view and edit the configuration properties for the realm to which Integration Server connects.

Note:
When the encoding type is IData, it is optional to include *_properties* in the provider filter. For example, if you want Universal Messaging to filter for messages where the contents of the *_properties/color* field is equal to "blue", the provider filter would be: color='blue'. However, when the encoding type is protocol buffers, you need to include *_properties* in the provider filter. For example, _properties.color='blue'. If you want a provider filter that operates on the contents of *_properties* to work regardless of the encoding type, always include *_properties* in the filter expression.

Examples of Universal Messaging Provider Filters for Use with Protocol Buffers

Following are examples of provider filters for protocol buffer encoded documents. The Universal Messaging server applies the filter when it receives an instance of a document in the trigger condition that contains the provider filter.

Filter	Evaluates to true when...
stringField = 'abc'	The value of *stringField* is "abc".

Filter	Evaluates to true when...
stringField < 'B'	The value of the *stringField* is less than B
stringField < '5'	The value of the *stringField* is less than 5.
stringField = '1'	The value of the *stringField* is "1".
document.stringField='abc'	In the Document field named *document*, the value of the *stringField* is "abc".
stringList[1] = 'b'	The value of the second element in *stringList* is "b"
documentList[0].stringField = 'a'	In the first element of the Document list named *documentList*, the value of *stringField* is "a".
stringField1 = 'a' and stringField2 = 'b'	The value of *stringField1* is "a" and the value of *stringField2* is "b".
stringField1 = 'a' or stringField2 = 'b'	The value of *stringField1* is "a" or the value of *stringField2* is "b".
stringField1 = stringField2	The value of *stringField1* is the same as the value of *stringField2*.
integerField = 5 Where *integerField* is an Object field with the Java wrapper type java.lang.Integer applied.	The value of *integerField* is "5".
integerField > 1 Where *integerField* is an Object field with the Java wrapper type java.lang.Integer applied.	The value of *integerField* is greater than 1.
boolean1 = true and boolean2 = false Where *boolean1* and *boolean2* are Object fields with the Java wrapper type java.lang.Boolean applied.	The value of *boolean1* is true and the value of *boolean2* is false.

For more information about server-side filtering on Universal Messaging, including syntax, see the Universal Messaging documentation.

Creating Filters for Use with webMethods Broker

If a webMethods messaging trigger subscribes to publishable document types associated with a webMethods Broker connection alias, you can specify a single filter that can be used by webMethods Broker and/or Integration Server. Use the **Filter** column in the Condition detail table to specify the filter.

The filter can be saved with the subscription on the webMethods Broker and with the webMethods messaging trigger on the Integration Server. This is because some filter syntax that is valid on Integration Server is not valid on webMethods Broker. For example, webMethods Broker prohibits the use of certain words or characters in field names, such as Java keywords, @, *, and names containing white spaces. The location of the filter and whether webMethods Broker and/or Integration Server applies the filter, depends on the filter syntax, which is evaluated at design time.

When you save a webMethods messaging trigger, Integration Server and webMethods Broker evaluate the filter in the **Filter** column.

- Integration Server evaluates the filter to make sure it uses the proper syntax. If the syntax is correct, Integration Server saves the webMethods messaging trigger in an enabled state. If the syntax is incorrect, Integration Server saves the webMethods messaging trigger in a disabled state.
- webMethods Broker evaluates the filter syntax to determine if the filter syntax is valid on the webMethods Broker. If webMethods Broker determines that the syntax is valid for the webMethods Broker, it saves the filter with the document type subscription. If the webMethods Broker determines that the filter syntax is not valid on the webMethods Broker or if attempting to save the filter on the webMethods Broker would cause an error, webMethods Broker saves the subscription without the filter.

  webMethods Broker saves as much of a filter as possible with the subscription. For example, suppose that a filter consists of more than one expression, and only one of the expressions contains the syntax that the webMethods Broker considers invalid. webMethods Broker saves the expressions it considers valid with the subscription on the webMethods Broker. (Integration Server saves all the expressions.)

When a filter is saved only on Integration Server and not on webMethods Broker, the performance of Integration Server can be affected. When the webMethods Broker applies the filter to incoming documents, it discards documents that do not meet filter criteria. Integration Server only receives documents that meet the filter criteria. If the subscription filter resides only on Integration Server, webMethods Broker automatically places the document in the subscriber's queue. webMethods Broker routes all the documents to the subscriber, creating greater network traffic between the webMethods Broker and the Integration Server and requiring more processing by the Integration Server.

Using Hints in Filters

Hints are used to further define a filter. You add hints to the end of a subscription in the **Filter** field.

Hints use the following syntax:

```
{hint: HintName=Value}
```

The table below identifies the HintNames that you can use with a document subscription.

Hint	Description
`IncludeDeliver`	When set to `true`, the filter applies to documents that are delivered to the client and documents that are delivered to the subscription queue. By default, filters are only applied to documents that are delivered to the subscription queue.
`LocalOnly`	When set to `true`, the filter is applied only to documents that originate from the webMethods Broker to which the Integration Server is connected. Documents originating from a different webMethods Broker are discarded.
`DeadLetterOnly`	When set to `true`, a deadletter subscription is created for the document type specified in the webMethods messaging trigger. The webMethods messaging trigger that subscribes to this hint receives messages that do not have subscribers. To learn more about detecting deadletters, see the *webMethods Broker Java Client API Reference*.

Keep the following points in mind when you add hints to filters:

- Hints must be added at the end of the filter string in the **Filter** field.
- Hints must be in the following format:

  ```
  {hint: HintName=Value}
  ```

  For example, the following filter will match only those documents that originate from the webMethods Broker to which the Integration Server is connected and the value of *city* is equal to `Fairfax`.

  ```
  %city% L_EQUALS "Fairfax" {hint:LocalOnly=true}
  ```

- A filter can also contain a combination of subscription hints. For example, the following filter will match only those documents that do not have a subscriber and that originate from the webMethods Broker to which the Integration Server is connected.

  ```
  {hint:DeadLetterOnly=true} {hint:LocalOnly=true}
  ```

Detecting Deadletters with Hints

A deadletter is an unclaimed published document. If there are no subscribers for a document that is published, the webMethods Broker returns an acknowledgement to the publisher and then discards the document. If, however, a deadletter subscription exists for the document, the webMethods Broker deposits the document in the queue containing the deadletter subscription.

A deadletter subscription allows you to trap unclaimed documents. Detecting and trapping deadletters is a valuable way to quickly identify and resolve discrepancies between a published document and the filtering criteria specified by a subscriber.

You create a deadletter subscription by inserting the `DeadLetterOnly` hint to the subscription filter. For more information about creating deadletter subscriptions using the `DeadLetterOnly` hint, see "Using Hints in Filters" on page 691.

When using the `DeadLetterOnly` hint, keep the following points in mind:

- If the `DeadLetterOnly` hint is used in a filter that contains other expressions, the other expressions are ignored by the webMethods Broker. Consider the following example:

  ```
  %city% L_EQUALS "Fairfax" {hint:DeadLetterOnly=true} {hint:LocalOnly=true}
  ```

  webMethods Broker will ignore the expression `%city% L_EQUALS "Fairfax"` in the filter and will trap only those documents that do not have a subscriber and that originate from the webMethods Broker to which the Integration Server is connected.

- If the filter is registered on Integration Server but not on the webMethods Broker, the `DeadLetterOnly` subscription traps only the documents that are rejected by Integration Server. The filter does not trap the documents rejected by the webMethods Broker unless the filter is registered on the webMethods messaging trigger.

- Both the `LocalOnly` and `IncludeDeliver` hints are implied.

Note:
If you are using Universal Messaging you can configure a dead events store. For more information, see the Universal Messaging documentation.

Using Multiple Conditions in a webMethods Messaging Trigger

You can build webMethods messaging triggers that can contain more than one condition. Each condition can associate one or more documents with a service. You can use the same service or different services for each condition. You can create only one join condition in a webMethods messaging trigger, but a webMethods messaging trigger can contain any number of simple conditions.

When a webMethods messaging trigger receives a document, Integration Server determines which service to invoke by evaluating the webMethods messaging trigger conditions. Integration Server evaluates the webMethods messaging trigger conditions in the same order in which the conditions appear in the editor. It is possible that a document could satisfy more than one condition in a webMethods messaging trigger. However, Integration Server executes only the service associated with the first satisfied condition and ignores the remaining conditions. Therefore, the order in which you list conditions is important.

When you build a webMethods messaging trigger with multiple conditions, each condition can specify the same service. However, you should avoid creating conditions that specify the same publishable document type. If the conditions in a webMethods messaging trigger specify the same publishable document type, Integration Server always executes the condition that appears first. For example, if a webMethods messaging trigger contained the following conditions:

Condition Name	Service	Document Types
ConditionAB	serviceAB	documentA or documentB

Condition Name	Service	Document Types
ConditionA	serviceA	documentA

Integration Server will never execute serviceA. Whenever Integration Server receives documentA, the document satisfies ConditionAB, and Integration Server executes serviceAB.

Using Multiple Conditions for Ordered Service Execution

You might create a webMethods messaging trigger with multiple conditions to handle a group of published documents that must be processed in a specific order. For each condition, associate one publishable document type with a service. Place your conditions in the order in which you want the services to execute. In the **Processing mode** property, specify serial document processing so that the webMethods messaging trigger will process the documents one at a time, in the order in which they are received. The serial dispatching ensures that the services that process the documents do not execute at the same time. (This assumes that the documents are published and therefore received in the proper order.)

Note:
Using multiple conditions to achieve ordered service execution is only supported for webMethods messaging triggers that receive messages from the webMethods Broker.

You might want to use multiple conditions to control the service execution when a service that processes a document depends on another service successfully executing. For example, to process a purchase order, you might create one service that adds a new customer record to a database, another that adds a customer order, and a third that bills the customer. The service that adds a customer order can only execute successfully if the new customer record has been added to the database. Likewise, the service that bills the customer can only execute successfully if the order has been added. You can ensure that the services execute in the necessary order by creating a webMethods messaging trigger that contains one condition for each expected publishable document type. You might create a webMethods messaging trigger with the following conditions:

Condition Name	Service	Document Types
Condition1	addCustomer	customerName
Condition2	addCustomerOrder	customerOrder
Condition3	billCustomer	customerBill

If you create one webMethods messaging trigger for each of these conditions, you could not guarantee that the Integration Server would invoke services in the required order even if publishing occurred in that order. Specifying serial dispatching for the webMethods messaging trigger ensures that a service will finish executing before the next document is processed. For example, Integration Server could still be executing addCustomer, when it receives the documents customerOrder and customerBill. If you specified concurrent dispatching instead of serial dispatching, the Integration Server might execute the services addCustomerOrder and billCustomer before it finished executing addCustomer. In that case, the addCustomerOrder and billCustomer services would fail.

Important:
An ordered scenario assumes that documents are published in the correct order and that you set up the webMethods messaging trigger to process documents serially. For more information about specifying the document processing for a webMethods messaging trigger, see "Selecting Message Processing" on page 711.

Ordering Conditions in a webMethods Messaging Trigger

The order in which you list conditions in the editor is important because it indicates the order in which the Integration Server evaluates the conditions at run time. When the Integration Server receives a document, it invokes the service specified in the first condition that is satisfied by the document. The remaining conditions are ignored.

To change the order of conditions in a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger.
2. Under **Conditions**, select the condition to be moved.
3. Click or to move the condition up or down.
4. Click **File > Save** to save the webMethods messaging trigger.

Disabling and Enabling a webMethods Messaging Trigger

You can use the **Enabled** property to disable or enable a webMethods messaging trigger. When you disable a webMethods messaging trigger, Integration Server disconnects the trigger client on the webMethods Broker. The webMethods Broker removes the document subscriptions created by the trigger client. The webMethods Broker does not place published documents in client queues for disabled webMethods messaging triggers. When you enable a disabled webMethods messaging trigger, Integration Server connects the trigger client to the webMethods Broker and re-establishes the document subscriptions on the webMethods Broker.

Note:
You can also suspend document retrieval and document processing for a webMethods messaging trigger. Unlike disabling a webMethods messaging trigger, suspending retrieval and processing does not destroy the client queue. The webMethods Broker continues to enqueue documents for suspended webMethods messaging triggers. However, Integration Server does not retrieve or process documents for suspended webMethods messaging triggers. For more information about suspending webMethods messaging triggers, see *webMethods Integration Server Administrator's Guide*.

You cannot disable a webMethods messaging trigger during trigger service execution.

To disable or enable a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger that you want to disable or enable.

2. In the Properties view, under **General**, set **Enabled** to one of the following:

Select...	To...
False	Disable the webMethods messaging trigger.
True	Enable the webMethods messaging trigger.

3. Click **File > Save**.

   You can enable only valid webMethods messaging triggers. If the webMethods messaging trigger is not valid, Designer displays an error message when you save the webMethods messaging trigger prompting you to cancel the save or continue the save with a disabled webMethods messaging trigger. For more information about requirements for a valid webMethods messaging trigger, see " webMethods Messaging Trigger Requirements" on page 681.

Disabling and Enabling a webMethods Messaging Trigger in a Cluster or Non-Clustered Group

When a webMethods messaging trigger exists on multiple Integration Servers in a cluster or in a non-clustered group, the subscriptions created by the webMethods messaging trigger remain active even if you disable the webMethods messaging trigger from one of the Integration Servers. This is because the client created for the webMethods messaging trigger on the webMethods Broker is shared. The client on the webMethods Broker becomes disconnected when you disable the webMethods messaging trigger on all the servers in the cluster or non-clustered group of servers. Even when the shared webMethods messaging trigger client becomes disconnected, the subscriptions established by the webMethods messaging trigger remain active. The webMethods Broker continues to enqueue documents for the webMethods messaging trigger. When you re-enable the webMethods messaging trigger on any server in the cluster or non-clustered group, all the queued documents that did not expire will be processed.

To disable a webMethods messaging trigger in a cluster or non-clustered group of Integration Servers, disable the webMethods messaging trigger on each Integration Server, and then manually remove the document subscriptions created by the webMethods messaging trigger from the webMethods Broker.

About Join Time-Outs

When you create a join condition using an All (AND) join or an Only one (XOR), you need to specify a join time-out. A *join time-out* specifies how long Integration Server waits for the other documents in the join condition. Integration Server uses the join time-out period to avoid deadlock situations (such as waiting for a document that never arrives) and to avoid duplicate service invocation.

How the join time-out affects document processing by the webMethods messaging trigger is different for each join type.

- For an All (AND) join, the join time-out determines how long Integration Server waits to receive an instance of each publishable document type in the condition.
- For an Only one (XOR) join, the join time-out determines how long Integration Server discards instances of publishable document types in the condition after it receives an instance document of one of the publishable document types.
- An Any (OR) join condition does not need a join time-out. Integration Server treats an Any (OR) join condition like a webMethods messaging trigger with multiple simple conditions that all use the same trigger service.

Join Time-Outs for All (AND) Join Conditions

A join time-out for an All (AND) join condition specifies how long the Integration Server waits for all of the documents specified in the join condition. When Integration Server pulls a document from the webMethods messaging trigger queue, it determines which condition the document satisfies. If the document satisfies an All (AND) join condition, the Integration Server starts the join time-out period and moves the document from the webMethods messaging trigger queue to the ISInternal database. Integration Server assigns the document a status of "pending." Integration Server then waits for the remaining documents in the join condition. Only documents with the same activation ID as the first received document will satisfy the join condition.

If Integration Server receives all of the documents specified in the join condition (and processes the documents from the trigger queue) before the time-out period elapses, it executes the service specified in the condition. If Integration Server does not receive all of the documents before the time-out period elapses, Integration Server removes the pending documents from the database and generates a journal log message.

When the time-out period elapses, the next document in the webMethods messaging trigger queue that satisfies the All (AND) condition causes the time-out period to start again. Integration Server places the document in the database and assigns a status of "pending" even if the document has the same activation ID as an earlier document that satisfied the join condition. Integration Server then waits for the remaining documents in the join condition.

Join Time-Outs for Only One (XOR) Join Conditions

A join time-out for an Only one (XOR) join condition specifies how long Integration Server discards instances of the other documents in the condition. When Integration Server pulls the document from the webMethods messaging trigger queue, it determines which condition the document satisfies. If that condition is an Only one (XOR) condition, the Integration Server executes the service specified in the condition. When it pulls the document from the webMethods messaging trigger queue, Integration Server starts the time-out period. For the duration of the time-out period, Integration Server discards any documents of the type specified in the join condition. Integration Server discards only those documents with same activation ID as the first document.

When the time-out period elapses, the next document in the webMethods messaging trigger queue that satisfies the Only one (XOR) condition causes the trigger service to execute and the time-out

period to start again. Integration Server executes the service even if the document has the same activation ID as an earlier document that satisfied the join condition. Integration Server generates a journal log message when the time-out period elapses for an Only one (XOR) condition.

Setting a Join Time-Out

When configuring webMethods messaging trigger properties, you can specify whether a join condition times out and if it does, what the time-out period should be. The time-out period indicates how long Integration Server waits for additional documents after receiving the first document specified in the join condition.

To set a join time-out

1. In the Package Navigator view of Designer, open the webMethods messaging trigger that you want to set the join time-out.

2. In the Properties view, under **General**, set **Join expires** to one of the following:

Select...	To...
True	Indicate that Integration Server stops waiting for the other documents in the join condition once the time-out period elapses. In the **Expire after** property, specify the length of the join time-out period. The default time period is 1 day.
False	Indicate that the join condition does not expire. Integration Server waits indefinitely for the additional documents specified in the join condition. Set the **Join expires** property to **False** only if you are confident that all of the documents will be received. **Important:** A join condition is persisted across server restarts. To remove a waiting join condition that does not expire, disable, then re-enable and save the webMethods messaging trigger. Re-enabling the webMethods messaging trigger effectively recreates the webMethods messaging trigger.

3. Click **File > Save** to save the webMethods messaging trigger.

About Priority Message Processing

Priority messaging determines the order in which a webMethods messaging trigger receives and subsequently processes the documents from the messaging provider. How priority messaging works depends on the messaging provider from which the trigger receives documents.

When priority messaging is enabled for a webMethods messaging trigger that receives documents from webMethods Broker, webMethods Broker places documents in the trigger client queue using the following criteria:

- **Message priority.** webMethods Broker inserts documents with the highest priority at the top of the client queue. A value of 0 is the lowest processing priority; a value of 9 indicates expedited processing.
- **Message publication time.** webMethods Broker orders documents with the same priority in the client queue according to the time at which the document was published. Within documents of the same priority, webMethods Broker inserts the most recently published document after the earlier documents. This ensures that the webMethods messaging trigger receives and processes the documents with the same priority in publication order.

Integration Server receives and processes the documents following the order in which the documents appear in the client queue on the webMethods Broker.

When priority messaging is enabled for a webMethods messaging trigger that receives documents from a Universal Messaging server, Universal Messaging and Integration Server adapt to expedite processing of higher priority documents over lower priority documents.

Note:
Priority messaging applies only to documents that are routed through the webMethods Broker and Universal Messaging. Priority messaging does not apply to locally published documents.

To use priority messaging, you configure both the publishing side and the subscribing side.

- On the publishing side, set a message priority level in the document envelope. The priority level indicates how quickly the document should be processed once it is published. A value of 0 is the lowest processing priority; a value of 9 indicates expedited processing. The default priority is 4.
- On the subscribing side, enable priority messaging for the webMethods messaging trigger. This is necessary only for triggers that receive documents from webMethods Broker.

  Note:
  All webMethods messaging triggers that receive documents from Universal Messaging receive and process documents in priority fashion. Priority messaging cannot be disabled for webMethods messaging triggers that receive documents from Universal Messaging.

Enabling and Disabling Priority Message Processing for a webMethods Messaging Trigger

When enabling or disabling priority message processing for a webMethods messaging trigger, keep the following points in mind:

- Priority messaging only applies to documents that are published to the webMethods Broker and Universal Messaging. It does not apply to locally published documents.
- All webMethods messaging triggers that receive documents from Universal Messaging receive and process documents in priority fashion. Priority messaging cannot be disabled for

webMethods messaging triggers that receive documents from Universal Messaging. That is, Integration Server ignores the value of the **Priority** property if the webMethods messaging trigger subscribes to one or more publishable document types associated with a Universal Messaging connection alias.

- When you enable or disable priority messaging for a webMethods messaging trigger that receives documents from webMethods Broker, Integration Server disconnects the trigger client on the webMethods Broker and recreates the associated webMethods messaging trigger client queue. Any documents that existed in the webMethods messaging trigger client queue before you enabled or disabled priority messaging will be lost. For this reason, you may want to ensure the webMethods messaging trigger queue on Integration Server is empty before enabling or disabling priority messaging.
- Priority messaging may consume webMethods Broker resources and can introduce latency into document processing by webMethods messaging triggers. For more information about how priority messaging may impact performance, refer to the *webMethods Broker Java Client API Reference*.

To enable or disable priority message processing for a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to enable priority processing of documents.
2. In the Properties view, under **General**, set **Priority** enabled to one of the following:

Select...	To...
True	Enable priority processing.
False	Disable priority processing.

3. Click **File > Save** to save the webMethods messaging trigger.

About Execution Users for webMethods Messaging Triggers

For a webMethods messaging trigger that receives documents from Universal Messaging, the **Execution user** property indicates which credentials Integration Server should use when invoking services associated with the trigger. When a client invokes a service via an HTTP request, Integration Server checks the credentials and user group membership of the client against the Execute ACL assigned to the service. Integration Server performs this check to make sure that the client is allowed to invoke that service. When a webMethods messaging trigger executes, however, Integration Server invokes the service when it receives a message rather than as a result of a client request. Integration Server does not associate user credentials with a message. You can specify which credentials Integration Server should supply when invoking a webMethods messaging trigger service after receiving a document from Universal Messaging by setting the **Execution user** property for the trigger.

You can instruct Integration Server to invoke a service using the credentials of one of the predefined user accounts (Administrator, Default, Developer, Replicator). You can also specify a user account that you or another server administrator defined. When webMethods messaging trigger receives a message from Universal Messaging, Integration Server uses the credentials for the specified user account to invoke the service specified in the trigger condition.

Note:
For a webMethods messaging trigger that receives locally published messages or messages from the webMethods Broker, Integration Server, uses the user account specified in the **Run Trigger Service As User** property on the **Settings > Resources > Store Settings** page in Integration Server Administrator. For more information about the **Run Trigger Service As User** property, see *webMethods Integration Server Administrator's Guide*.

Assigning an Execution User to a webMethods Messaging Trigger

Make sure that the user account you select includes the credentials required by the execute ACL assigned to the services associated with the webMethods messaging trigger.

Note:
The **Execution user** property only applies to webMethods messaging triggers that receive documents from Universal Messaging. The publishable document type to which a trigger subscribes determine the messaging provider from which the trigger receives documents. The **Execution user** property is display only if a webMethods messaging trigger receives locally published documents or documents published to webMethods Broker.

To assign an execution user for a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to set the execution user.

2. In the Properties view, under **General**, in the **Execution user** property, type the name of the user account whose credentials Integration Server uses to execute a service associated with the webMethods messaging trigger. You can specify a locally defined user account or a user account defined in a central or external directory.

3. Click **File > Save**.

About Capacity and Refill Level for the webMethods Messaging Trigger Queue

On Integration Server, the webMethods messaging trigger queue contains documents waiting for processing. Integration Server assigns each webMethods messaging trigger a queue. A document remains in the webMethods messaging trigger queue until it is processed by the webMethods messaging trigger.

You can determine the capacity of each webMethods messaging trigger queue. The capacity indicates the maximum number of documents that Integration Server can store for that webMethods messaging trigger.

For a webMethods messaging trigger that receives documents from the webMethods Broker, you can also specify a refill level to indicate when Integration Server should retrieve more documents for the webMethods messaging trigger. The difference between the capacity and the refill level determines up to how many documents the Integration Server retrieves for the webMethods messaging trigger from the webMethods Broker. For example, if you assign the webMethods messaging trigger queue a capacity of 10 and a refill level of 4, the Integration Server initially retrieves 10 documents for the webMethods messaging trigger. When only 4 documents remain to be processed in the webMethods messaging trigger queue, Integration Server retrieves up to 6 more documents. If 6 documents are not available, Integration Server retrieves as many as possible.

For a webMethods messaging trigger that receives documents from the webMethods Broker, the capacity and refill level also determine how frequently Integration Server retrieves documents for the webMethods messaging trigger and the combined size of the retrieved documents, specifically:

- The greater the difference between capacity and refill level, the less frequently Integration Server retrieves documents from the webMethods Broker. However, the combined size of the retrieved documents will be larger.
- The smaller the difference between capacity and refill level, the more frequently Integration Server retrieves documents. However, the combined size of the retrieved documents will be smaller.

Note:
A refill level can be set for webMethods messaging triggers that receive documents from the webMethods Broker only. Refill level does not apply to webMethods messaging triggers that receive documents from Universal Messaging.

For a webMethods messaging trigger that receives messages from Universal Messaging, Integration Server receives documents for the trigger one at a time until the trigger queue is at capacity. After the number of documents in the trigger queue equals the configured capacity, Integration Server stops receiving documents. When the number of documents awaiting processing in the trigger queue is less than the configured capacity, the trigger resumes receiving messages from Universal Messaging.

Guidelines for Setting Capacity and Refill Levels for webMethods Messaging Triggers

When you set values for capacity and refill level, you need to balance the frequency of document retrieval with the combined size of the retrieved documents. Use the following guidelines to set values for capacity and refill level for a webMethods messaging trigger that retrieves messages from webMethods Broker.

- If the webMethods messaging trigger subscribes to small documents, set a high capacity. Then, set refill level to be 30% to 40% of the capacity. Integration Server retrieves documents for this webMethods messaging trigger less frequently, however, the small size of the documents indicates that the combined size of the retrieved documents will be manageable. Additionally,

setting the refill level to 30% to 40% ensures that the webMethods messaging trigger queue does not empty before the Integration Server retrieves more documents. This can improve performance for high-volume and high-speed processing.

- If the webMethods messaging trigger subscribes to large documents, set a low capacity. Then, set the refill level to just below slightly less than the capacity. Integration Server retrieves documents more frequently, however, the combined size of the retrieved documents will be manageable and will not overwhelm Integration Server.

Setting Capacity and Refill Level for a webMethods Messaging Trigger

To set trigger queue capacity and refill level

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to specify the webMethods messaging trigger queue capacity.

2. In the Properties view, under **Trigger queue**, in the **Capacity** property, specify the maximum number of documents that the trigger queue can contain. The default is 10.

3. In the **Refill level** property, specify the number of unprocessed documents that must remain in this webMethods messaging trigger queue before Integration Server retrieves more documents for the queue from the webMethods Broker. The default is 4.

   The **Refill level** value must be less than or equal to the **Capacity** value.

   Note:
   The **Refill level** property applies to webMethods messaging triggers that receive documents from the webMethods Broker only. The **Refill level** property does not apply to webMethods messaging triggers that receive documents from Universal Messaging.

4. Click **File > Save** to save the webMethods messaging trigger.

Notes:

- The server administrator can use the Integration Server Administrator to gradually decrease the capacity and refill levels of all webMethods messaging trigger queues. The server administrator can also use the Integration Server Administrator to change the **Capacity** or **Refill leve**l values for a webMethods messaging trigger. For more information, see *webMethods Integration Server Administrator's Guide*.

- You can specify whether Integration Server should reject locally published documents when the queue for the subscribing webMethods messaging trigger is at maximum capacity. For more information about this feature, see the description for the watt.server.publish.local.rejectOOS parameter in *webMethods Integration Server Administrator's Guide*.

About Document Acknowledgements for a webMethods Messaging Trigger

When a trigger service finishes processing a guaranteed document, Integration Server returns an acknowledgement to the messaging provider. Upon receipt of the acknowledgement, the messaging provider removes its copy of the document from storage. By default, Integration Server returns an acknowledgement for a guaranteed document as soon as it finishes processing the document.

Note: Integration Server returns acknowledgements for guaranteed documents only. Integration Server does not return acknowledgements for volatile documents.

You can increase the number of document acknowledgements returned at one time by changing the value of the **Acknowledgement queue size** property. The acknowledgement queue is a queue that contains pending acknowledgements for guaranteed documents processed by the webMethods messaging trigger. When the acknowledgement queue size is greater than one, a server thread places a document acknowledgement into the acknowledgement queue after it finishes executing the trigger service. Acknowledgements collect in the queue until a background thread returns them as a group to the sending resource.

If the **Acknowledgement queue size** is set to one, acknowledgements will not collect in the acknowledgement queue. Instead, Integration Server returns an acknowledgement to the sending resource immediately after the trigger service finishes executing.

If a resource or connection failure occurs before acknowledgements are sent or processed, the transport redelivers the previously processed, but unacknowledged documents. The number of documents redelivered to a webMethods messaging trigger depends on the size on the number of guaranteed documents that were processed but not acknowledged before failure occurred. If exactly-once processing is configured for the webMethods messaging trigger, Integration Server detects the redelivered, guaranteed documents as duplicates and discards them without re-processing them. For more information about exactly-once processing, see "Exactly-Once Processing for webMethods Messaging Triggers" on page 723.

Increasing the size of a webMethods messaging trigger's acknowledgement queue can provide the following benefits:

- **Reduces network traffic.** Returning acknowledgements one at a time for each guaranteed document that is processed can result in a high volume of network traffic. Configuring the webMethods messaging trigger so that Integration Server returns several document acknowledgements at once can reduce the amount of network traffic.
- **Increases server thread availability.** If the size of the acknowledgement queue is set to 1 (the default), Integration Server releases the server thread used to process the document only after returning the acknowledgement. If the size of the acknowledgement queue is greater than 1, Integration Server releases the server thread used to process the document immediately after the thread places the acknowledgement into the acknowledgement queue. When acknowledgements collect in the queue, server threads can be returned to the thread pool more quickly.

Setting the Size of the Acknowledgement Queue

To set the size of the acknowledgement queue for a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to specify the size of the acknowledgement queue.

2. In the Properties view, under **Trigger queue**, in the **Acknowledgement queue size** property, specify the maximum number of pending document acknowledgements for the webMethods messaging trigger.

   The value must be greater than zero. The default is 1.

3. Click **File > Save** to save the webMethods messaging trigger.

About Message Processing

Message processing determines the order in which Integration Server processes the documents received by a webMethods messaging trigger.

- In serial processing, Integration Server processes the documents received by a webMethods messaging trigger one after the other, in the order in which the documents were received.
- In concurrent processing, Integration Server processes the documents by a webMethods messaging trigger in parallel.

Serial Processing

In serial processing, Integration Server processes the documents received by a webMethods messaging trigger one after the other. Integration Server retrieves the first document received by the webMethods messaging trigger, determines which condition the document satisfies, and executes the service specified in the webMethods messaging trigger condition. Integration Server waits for the service to finish executing before retrieving the next document received by the webMethods messaging trigger.

In serial processing, Integration Server processes documents for the webMethods messaging trigger in the same order in which it retrieves the documents from the messaging provider. However, Integration Server processes documents for a serial trigger more slowly than it processes documents for a concurrent trigger.

If your webMethods messaging trigger contains multiple conditions to handle a group of published documents that must be processed in a specific order, use serial processing. This is sometimes called ordered service execution. Only triggers that receive messages from webMethods Broker can perform ordered service execution.

When a webMethods messaging trigger receives documents from webMethods Broker, the queue for the serial trigger on the webMethods Broker has a Shared Document Order mode of “Publisher”.

When a webMethods messaging trigger receives documents from Universal Messaging, the named object for a trigger with serial processing is a priority named object. That is, in Universal Messaging Enterprise Manager, the named object for the trigger has the **Subscription Priority** check box selected.

Serial Processing in a Cluster or Non-Clustered Group of Integration Servers

Serial document processing determines how the messaging provider distributes guaranteed documents to the individual servers within a cluster or non-clustered group. In a cluster or non-clustered group, the individual Integration Servers share the same client. For example, if the messaging provider is the webMethods Broker, the servers act as a single webMethods Broker client and share the same trigger client queues and document subscriptions. With serial processing, servers in a cluster or non-clustered group can process documents from a publisher in the same order in which the documents were published.

Note:
In addition to the term "non-clustered group," the terms "stateless cluster" and "external cluster" are sometimes used to describe the situation in which a group of Integration Servers function in a manner similar to a cluster but are not part of a configured cluster.

For each webMethods messaging trigger, each server in the cluster or non-clustered group maintains a trigger queue in memory. This allows multiple servers to process documents for a single webMethods messaging trigger. The messaging provider manages the distribution of documents to the individual webMethods messaging triggers in the cluster or non-clustered group.

How the messaging provider distributes documents for a serial trigger on the Integration Servers in the cluster or group to ensure that documents from a single publisher are processed in publication order varies:

- webMethods Broker distributes documents so that the Integration Servers in the cluster or non-clustered group process guaranteed documents from a single publisher in the same order in which the documents were published. Multiple Integration Servers can process documents for a single trigger, but only one Integration Server in the cluster or non-clustered group processes documents for a particular publisher. For more information, see "Serial Processing with the webMethods Broker in a Clustered or a Non-Clustered Group of Integration Servers" on page 707
- Universal Messaging distributes all the documents to which a particular serial trigger subscribes to the same Integration Server in a cluster or non-clustered group. Regardless of the document publisher, all of the published documents to which a specific serial trigger subscribes are received and processed by the same Integration Server. Because a serial trigger processes only one document at a time, this distribution approach ensures that documents are processed in the same order in which they were published. For more information, see "Serial Processing with Universal Messaging in a Clustered or a Non-Clustered Group of Integration Servers" on page 709.

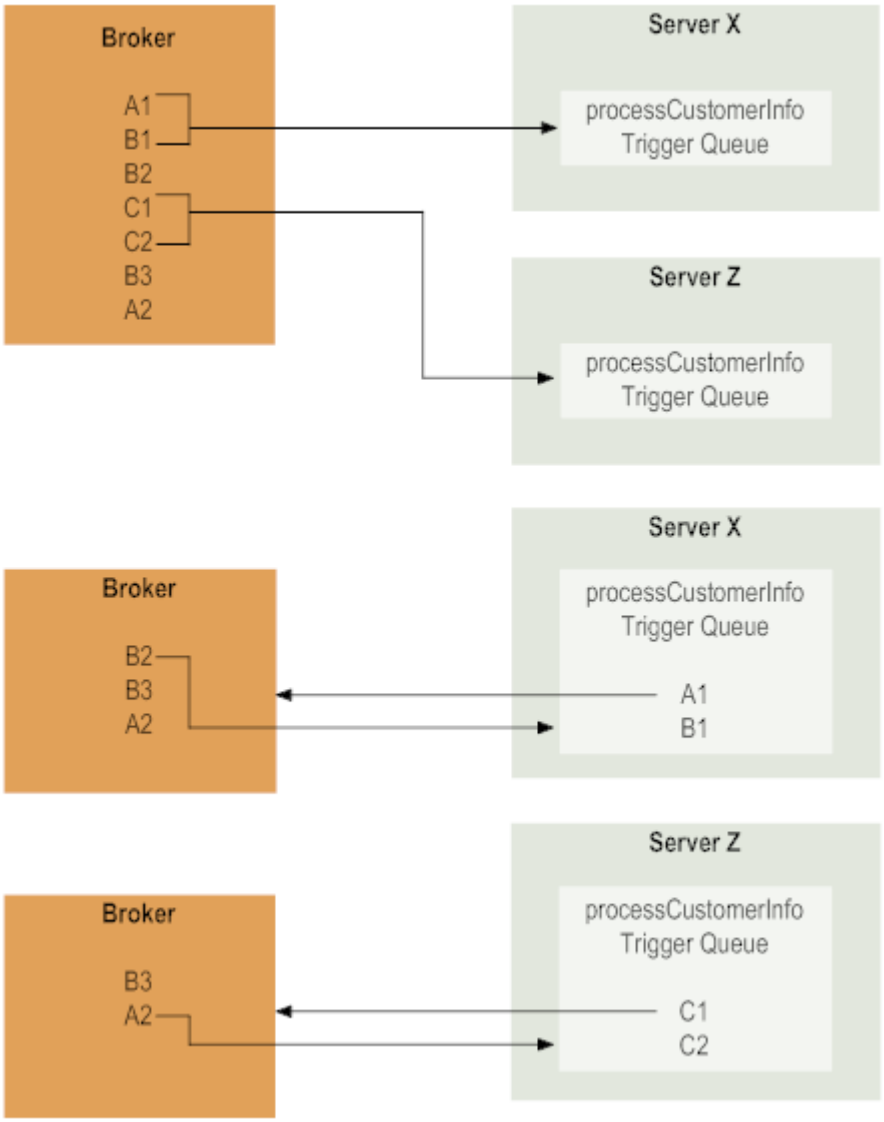
Serial Processing with the webMethods Broker in a Clustered or a Non-Clustered Group of Integration Servers

To ensure that a serial webMethods messaging trigger processes guaranteed documents from individual publishers in publication order, the webMethods Broker distributes documents from one publisher to a single server in a cluster or non-clustered group. The webMethods Broker continues distributing documents from the publisher to the same server as long as the server contains unacknowledged documents from that publisher in the trigger queue. Once the server acknowledges all of the documents from the publisher to the webMethods Broker, other servers in the cluster or non-clustered group can process future documents from the publisher.

For example, suppose that a cluster contains two servers: ServerX and ServerZ. Each of these servers contains the webMethods messaging trigger processCustomerInfo. The processCustomerInfo webMethods messaging trigger specifies serial document processing with a capacity of 2 and a refill level of 1. For each publisher, the cluster must process documents for this webMethods messaging trigger in the publication order. In this example, the processCustomerInfo trigger client queue on the webMethods Broker contains documents from PublisherA, PublisherB, and PublisherC. PublisherA published documents A1 and A2, PublisherB published documents B1, B2, and B3, and PublisherC published documents C1and C2.

The following illustration and explanation describe how serial document processing works in a clustered environment that uses webMethods Broker as the messaging provider.

Serial processing in a cluster of Integration Servers



Step	Description
1	ServerX retrieves the first two documents in the queue (documents A1 and B1) to fill its processCustomerInfo trigger queue to capacity. ServerX begins processing document A1.
2	ServerZ retrieves the documents C1 and C2 to fill its processCustomerInfo trigger queue to capacity. ServerZ begins processing the document C1. Even though document B2 is the next document in the queue, the webMethods Broker does not distribute document B2 from PublisherB to ServerZ because ServerX contains unacknowledged documents from PublisherB.
3	ServerX finishes processing document A1 and acknowledges document A1 to the webMethods Broker.

Step	Description
4	ServerX requests 1 more document from the webMethods Broker. (The processCustomerInfo webMethods messaging trigger has refill level of 1.) The webMethods Broker distributes document B2 from PublisherB to ServerX.
5	ServerZ finishes processing document C1 and acknowledges document C1to the webMethods Broker
6	ServerZ requests 1 more document from the webMethods Broker. The webMethods Broker distributes document A2to ServerZ. ServerZ can process a document from PublisherA because the other server in the cluster (ServerX) does not have any unacknowledged documents from PublisherA. Even though document B3 is the next document in the queue, the webMethods Broker does not distribute document B3to ServerZ because ServerX contains unacknowledged documents from PublisherB.

Notes:

- The webMethods Broker and Integration Servers in a cluster cannot ensure that serial webMethods messaging triggers process volatile documents from the same publisher in the order in which the documents were published.
- When documents are delivered to the default client in a cluster, the webMethods Broker and Integration Servers cannot ensure that documents from the same publisher are processed in publication order. This is because the Integration Server acknowledges documents delivered to the default client as soon as they are retrieved from the webMethods Broker.

Serial Processing with Universal Messaging in a Clustered or a Non-Clustered Group of Integration Servers

To provide processing in publishing order for a serial trigger in a cluster or a non-clustered group of Integration Server, Universal Messaging routes all of the documents to which a serial webMethods messaging trigger subscribes to the same Integration Server. Because a serial trigger processes only one document at a time, this routing approach ensures that documents from the same publisher are processed in the order in which the documents were published.

To indicate that all of the documents for a serial trigger be sent to the same Integration Server, Integration Server creates a priority named object on Universal Messaging that corresponds to the serial trigger,. In Universal Messaging Enterprise Manager, the named object for the trigger has the **Subscription Priority** check box selected. With a priority named object, multiple consumers can connect to the named object but only one consumer is active. The active consumer has priority over the other consumers, which remain in fail-over mode. If the active consumer disconnects, one of the fail-over consumers becomes the active consumer and begins receiving documents. When a particular webMethods messaging trigger runs on multiple Integration Servers, each instance of the trigger is a consumer. Each trigger instance can connect to the priority named object but only one trigger at a time processes messages.

Note:

If you do not need serial processing of documents by publisher, but you want a trigger to process documents one at a time, select concurrent processing and set **Max execution threads** to 1. This configuration allows the trigger on each Integration Server in the cluster or group to process one document at a time.

Serial Triggers Migrated to Integration Server 9.9 or Later from 9.8 or Earlier

Prior toIntegration Server 9.9, when using Universal Messaging as the messaging provider, a webMethods messaging trigger with serial processing corresponded to a shared named object on Universal Messaging. As of Integration Server 9.9, a webMethods messaging trigger with serial processing corresponds to a priority named object on Universal Messaging. All webMethods messaging triggers created on Integration Server 9.9 or later will correspond to a priority named object. However, migrated serial triggers will still correspond to a shared named object. The trigger and named object will be out of sync. To synchronize the migrated serial trigger and the named object, you must do one of the following:

- If you are using a fresh install of Universal Messaging 9.9 or later (that is, the Universal Messaging server was not migrated), when you start Integration Server, synchronize the publishable document types with the provider using Designer or the built-in service pub.publish:syncToProvider. Synchronizing the publishable document types causes Integration Server to reload the webMethods messaging triggers. Integration Servercreates a priority named object for each serial trigger.
- If you are using an installation of Universal Messaging 9.9 or later that was migrated from an earlier version, you must delete and recreate the named object. For more information about deleting and recreating a named object associated with a trigger, see “Synchronizing the webMethods Messaging Trigger and Named Object on Universal Messaging ” on page 713.

Concurrent Processing

In concurrent processing, Integration Server processes the documents received by a webMethods messaging trigger in parallel. Integration Server processes as many documents in the webMethods messaging trigger queue as it can at the same time. Integration Server does not wait for the service specified in the webMethods messaging trigger condition to finish executing before it begins processing the next document in the trigger queue. You can specify the maximum number of documents Integration Server can process concurrently.

Concurrent processing provides faster performance than serial processing. The Integration Server process the documents in the trigger queue more quickly because the Integration Server can process more than one document at a time. However, the more documents Integration Server processes concurrently, the more server threads Integration Server dispatches, and the more memory the document processing consumes.

Additionally, for concurrent webMethods messaging triggers, Integration Server does not guarantee that documents are processed in the order in which they are received.

Concurrent document processing is equivalent to the Shared Document Order mode of “None” on the webMethods Broker.

When receiving messages from Universal Messaging, the Universal Messaging window size limits the number of documents that can be processed at one time by an individual trigger. By default, the window size of a client queue for the trigger is set to the sum of the **Capacity** and **Max execution threads** properties plus 5. For example, if the **Capacity** property is set to 10 and **Max execution threads** is set to 5, the client queue window size is 20. The window size set for a trigger overrides the default value specified in Universal Messaging.

Selecting Message Processing

To select message processing

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to specify the message processing mode.

2. In the Properties view, under **Message processing**, select one of the following for **Processing mode**:

Select...	To...
Serial	Specify that Integration Server should process documents received by the webMethods messaging trigger one after the other.
Concurrent	Specify that Integration Server should process as many documents received by the webMethods messaging trigger as it can at once.

3. If you selected concurrent processing, in the **Max execution threads** property, specify the maximum number of documents that Integration Server can process concurrently. Integration Server uses one server thread to process each document in the trigger queue.

4. If you selected serial processing and you want Integration Server to suspend document processing and document retrieval automatically when a trigger service ends with an error, under **Fatal error handling**, select **True** for the **Suspend on error** property.

   For more information about fatal error handling, see "Fatal Error Handling for a webMethods Messaging Trigger " on page 714.

5. Click **File > Save** to save the webMethods messaging trigger.

Notes:

- If you selected serial processing, Integration Server creates a priority named object on the channels that correspond to the publishable document types to which the trigger subscribes.
- If you selected concurrent processing, Integration Server creates a shared named object on the channels that correspond to the publishable document types to which the trigger subscribes

- Integration Server Administrator can be used to change the number of concurrent execution threads for a webMethods messaging trigger temporarily or permanently. For more information, see *webMethods Integration Server Administrator's Guide*.

Changing Message Processing When webMethods Broker Is the Messaging Provider

After you perform capacity planning and testing for your integration solution, you might want to modify the processing mode for a webMethods messaging trigger. Keep the following points in mind before you change the processing mode for a webMethods messaging trigger that receives documents from webMethods Broker:

- When you change the processing mode for a webMethods messaging trigger, Integration Server recreates the associated trigger client queue on the webMethods Broker.

  Important:
  Any documents that existed in the trigger client queue before you changed the message process mode will be lost.

- If you created the webMethods messaging trigger on an Integration Server connected to a configured webMethods Broker, you can only change the processing mode if Integration Server is currently connected to the webMethods Broker.
- If you change the document processing mode when Integration Server is not connected to the configured webMethods Broker, Designer displays a message stating that the operation cannot be completed.
- Integration Server does not change the processing mode if the webMethods Broker connection alias shares a client prefix.

  Note:
  A webMethods Broker connection alias shares a client prefix if the **Shared Client Prefix** property for the connection alias is set to **Yes**.

Changing Message Processing When Universal Messaging Is the Messaging Provider

You can change the message processing after you create a webMethods messaging trigger. For example, capacity planning might indicate that a concurrent trigger should be changed to serial. Keep the following points in mind when changing the processing mode from serial to concurrent or vice versa for a webMethods messaging trigger that receives documents from Universal Messaging:

- When you change the processing mode for a webMethods messaging trigger that uses a Universal Messaging connection alias that does not share a client prefix, Integration Server deletes and recreates the named object that corresponds to the trigger on Universal Messaging. The trigger and its associated named object on Universal Messaging remain in sync.

  Note:

A Universal Messaging connection alias does not share a client prefix if the **Shared Client Prefix** property for the connection alias is set to **No**.

- When you change the processing mode for a webMethods messaging trigger that uses a Universal Messaging connection alias that shares a client prefix, Integration Server *does not* delete and recreate the named object that corresponds to the trigger on Universal Messaging. As a result, the trigger on Integration Server will be out of sync with the associated named object on Universal Messaging. If the same trigger exists on other Integration Server, such as in a cluster or a non-clustered group of Integration Server, the changed trigger will also be out of sync with the trigger on other Integration Server. This affects document processing. One of the following situations occurs:

  - If you changed the processing mode from serial to concurrent, the corresponding named object on Universal Messaging remains a priority named object. The trigger continues to process documents concurrently, However, if the trigger exists on more than one Integration Server, such as in a cluster or a non-clustered group of Integration Servers, Universal Messaging distributes documents to the trigger on the first Integration Server to connect to Universal Messaging only. This trigger has priority and will receive and process all the documents to which the trigger subscribes. The other Integration Servers are connected to Universal Messaging but are in fail-over mode and will not receive or process documents unless the first trigger disconnects.

  - If you changed the processing mode from concurrent to serial, the corresponding named object on Universal Messaging remains a shared named object. Integration Server does not change the named object to be a priority named object. Consequently, if the trigger exists on more than one Integration Server, such as in a cluster or a non-clustered group of Integration Servers, Universal Messaging distributes documents to the trigger on each connected Integration Server. Universal Messaging does not distribute documents in a way that ensures that processing order matches publication order.

  For information about how to synchronize the trigger and the named object when the processing mode is out of sync, see "Synchronizing the webMethods Messaging Trigger and Named Object on Universal Messaging " on page 713.

  Note:
  A Universal Messaging connection alias shares a client prefix if the **Shared Client Prefix** property for the connection alias is set to **Yes**.

- Software AG does not recommend changing the processing mode for a trigger when more than one Integration Server connects to the same named object that corresponds to the trigger. For example, if the trigger is on an Integration Server that is part of a cluster or a non-clustered group, more than one Integration Server can share the same named object.

Synchronizing the webMethods Messaging Trigger and Named Object on Universal Messaging

A webMethods messaging trigger and the associated named object on Universal Messaging can get out of sync. For example, when you change the processing mode for a webMethods messaging trigger a webMethods messaging trigger that uses a Universal Messaging connection alias that shares a client prefix, Integration Server *does not* delete and recreate the named object that

corresponds to the trigger on Universal Messaging. As a result, the trigger on Integration Server is out of sync with the named object on Universal Messaging. To synchronize the webMethods messaging trigger and the associated named object, you must delete and recreate the named object.

Note:
A webMethods messaging trigger with a serial processing mode corresponds to a *priority* named object on Universal Messaging. A webMethods messaging trigger with a concurrent processing mode corresponds to a *shared* named object on Universal Messaging.

To synchronize the webMethods messaging trigger and the named object on Universal Messaging

- Do one of the following:
  - If the webMethods messaging trigger resides on the only Integration Server connected to Universal Messaging and the **Shared Client Prefix** property for the Universal Messagingconnection alias is set to **No**, start the trigger to delete and recreate the corresponding named object. You can start a trigger by disabling and then enabling the Universal Messagingconnection alias used by the trigger.

    Note: Integration Server starts triggers upon server restart.
  - If more than one Integration Server connects to Universal Messaging or the **Shared Client Prefix** property for the Universal Messaging connection alias is set to **Yes**, you must use Universal Messaging Enterprise Manager to delete the named object. Make sure to delete the named object when the named object is fully drained and no new documents will be sent to it. You many need to quiesce document publishers before deleting the named object. Then create the named object for the trigger by disabling and then enabling the Universal Messaging connection alias used by the trigger.

Fatal Error Handling for a webMethods Messaging Trigger

If a webMethods messaging trigger processes documents serially, you can configure fatal error handling for the webMethods messaging trigger. A fatal error occurs when the trigger service ends because of an exception. You can specify that Integration Server suspend the webMethods messaging trigger automatically if a fatal error occurs during trigger service execution. Specifically, Integration Server suspends document retrieval and document processing for the webMethods messaging trigger if the associated trigger service ends because of an exception.

When Integration Server suspends document processing and document retrieval for a webMethods messaging trigger, Integration Server writes the following message to the journal log:

```
Serial trigger triggerName has been automatically suspended due
to an exception.
```

Document processing and document retrieval remain suspended until one of the following occurs:

- You specifically resume document retrieval or document processing for the webMethods messaging trigger. You can resume document retrieval and document processing using Integration Server Administrator, built-in services (pub.trigger:resumeProcessing or

pub.trigger:resumeRetrieval), or by calling methods in the Java API (com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade.setProcessingSuspended() and com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade.setRetrievalSuspended()).

- Integration Server restarts, the webMethods messaging trigger is enabled or disabled (and then re-enabled), the package containing the webMethods messaging trigger reloads. (When Integration Server suspends document retrieval and document processing for a webMethods messaging trigger because of an error, Integration Server considers the change to be temporary. For more information about temporary vs. permanent state changes for webMethods messaging triggers, see *webMethods Integration Server Administrator's Guide*.)

For more information about resuming document processing and document retrieval, see *webMethods Integration Server Administrator's Guide* and the *webMethods Integration Server Built-In Services Reference*.

Automatic suspension of document retrieval and processing can be especially useful for serial webMethods messaging triggers that are designed to process a group of documents in a particular order. If the trigger service ends in error while processing the first document, you might not want to the webMethods messaging trigger to proceed with processing the subsequent documents in the group. If Integration Server automatically suspends document processing, you have an opportunity to determine why the trigger service did not execute successfully and then resubmit the document using webMethods Monitor.

By automatically suspending document retrieval as well, Integration Server prevents the webMethods messaging trigger from retrieving more documents. Because Integration Server already suspended document processing, new documents would just sit in the trigger queue. If Integration Server does not retrieve more documents for the webMethods messaging trigger and Integration Server is in a cluster, the documents might be processed more quickly by another Integration Server in the cluster.

Configuring Fatal Error Handling for a webMethods Messaging Trigger

Keep the following points in mind when configuring fatal error handling for a webMethods messaging trigger.

- You can configure fatal error handling for serial webMethods messaging triggers only.
- Integration Server does not automatically suspend webMethods messaging triggers because of transient errors that occur during trigger service execution. For more information about transient error handling, see "About Transient Error Handling for a webMethods Messaging Trigger " on page 716.

To configure fatal error handling for a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to specify the fatal error handling setting.

2. In the Properties view, under **Fatal error handling**, set the **Suspend on error** property to **True** if you want Integration Server to suspend document processing and document retrieval automatically when a trigger service ends with an error. Otherwise, select **False**. The default is **False**.

3. Click **File > Save** to save the webMethods messaging trigger.

About Transient Error Handling for a webMethods Messaging Trigger

When building a webMethods messaging trigger, you can specify whether or not Integration Server retries a trigger service when the trigger service fails because of a transient error caused by a run-time exception.

A *run-time exception* (specifically, an ISRuntimeException) occurs when the trigger service catches and wraps a transient error and then reissues it as an ISRuntimeException. A *transient error* is an error that arises from a temporary condition that might be resolved or corrected quickly, such as the unavailability of a resource due to network issues or failure to connect to a database. Because the condition that caused the trigger service to fail is temporary, the trigger service might execute successfully if the Integration Server waits and then re-executes the service.

You can configure transient error handling for a webMethods messaging trigger to instruct Integration Server to wait an specified time interval and then re-execute a trigger service automatically when an ISRuntimeException occurs. Integration Server re-executes the trigger service using the original input document.

When you configure transient error handling for a webMethods messaging trigger, you specify the following retry behavior:

- Whether Integration Server should retry trigger services for the webMethods messaging trigger. Keep in mind that a trigger service can retry only if it is coded to throw ISRuntimeExceptions.
- The maximum number of retry attempts Integration Server should make for each trigger service.
- The time interval between retry attempts.
- How to handle a retry failure. That is, you can specify what action Integration Server takes if all the retry attempts are made and the trigger service still fails because of an ISRuntimeException.

You can also configure Integration Server and/or a webMethods messaging trigger to handle transient errors that occur during trigger preprocessing. The trigger preprocessing phase encompasses the time from when a trigger first receives a message from it's local queue on webMethods messaging trigger to the time the trigger service executes.

For more information about transient error handling for trigger preprocessing, see "Transient Error Handling During Trigger Preprocessing" on page 737.

Service Requirements for Retrying a Trigger Service for a webMethods Messaging Trigger

To be eligible for retry, the trigger service must do one of the following to catch a transient error and reissue it as an ISRuntimeException:

- If the trigger service is a flow service, the trigger service must invoke pub.flow:throwExceptionForRetry. For more information about the pub.flow:throwExceptionForRetry, see the *webMethods Integration Server Built-In Services Reference*.
- If the trigger service is written in Java, the service can use com.wm.app.b2b.server.ISRuntimeException(). For more information about constructing ISRuntimeExceptions in Java services, see the *webMethods Integration Server Java API Reference* for the com.wm.app.b2b.server.ISRuntimeException class.

If a transient error occurs and the trigger service does not use pub.flow:throwExceptionForRetry or ISRuntimeException() to catch the error and throw an ISRuntimeException, the trigger service ends in error. Integration Server will not retry the trigger service.

Adapter services built on Integration Server 6.0 or later, and based on the ART framework, detect and propagate exceptions that signal a retry if a transient error is detected on their back-end resource. This behavior allows for the automatic retry when the service functions as a trigger service.

Note: Integration Server does not retry a trigger service that fails because a ServiceException occurred. A ServiceException indicates that there is something functionally wrong with the service. A service can throw a ServiceException using the EXIT step.

Handling Retry Failure

Retry failure occurs when Integration Server makes the maximum number of retry attempts and the trigger service still fails because of an ISRuntimeException. When you configure retry properties, you can specify one of the following actions to determine how Integration Server handles retry failure for a webMethods messaging trigger.

- **Throw exception.** When Integration Server exhausts the maximum number of retry attempts, Integration Server treats the last trigger service failure as a service error. This is the default behavior.
- **Suspend and retry later.** When Integration Server reaches the maximum number of retry attempts, Integration Server suspends the webMethods messaging trigger and then retries the trigger service at a later time.

Overview of Throw Exception for Retry Failure

Throwing an exception when retry failure occurs allows the webMethods messaging trigger to continue with document processing when retry failure occurs for a trigger service. You can configure audit logging in such a way that you can use webMethods Monitor to submit the document at a later time (ideally, after the condition that caused the transient error has been remedied).

The following table provides an overview of how Integration Server handles retry failure when the **Throw exception** option is selected.

Step	Description
1	Integration Server makes the final retry attempt and the trigger service fails because of an ISRuntimeException.
2	Integration Server treats the last trigger service failure as a service exception.
3	Integration Server rejects the document. If the document is guaranteed, Integration Server returns an acknowledgement to the webMethods Broker. If a trigger service generates audit data on error and includes a copy of the input pipeline in the service log, you can use webMethods Monitor to re-invoke the trigger service manually at a later time. Note that when you use webMethods Monitor to process the document, it is processed out of order. That is, the document is not processed in the same order in which it was received (or published) because the document was acknowledged to its transport when the retry failure occurred.
4	Integration Server processes the next document in the webMethods messaging trigger queue.

Overview of Suspend and Retry Later for Retry Failure

Suspending a webMethods messaging trigger and retrying the trigger service later when retry failure occurs provides a way to resubmit the document programmatically. It also prevents the webMethods messaging trigger from retrieving and processing other documents until the cause of the transient error condition has been remedied. This preserves the publishing order, which can be especially important for serial webMethods messaging triggers.

The following table provides more information about how the **Suspend and retry later** option works.

Step	Description
1	Integration Server makes the final retry attempt and the trigger service fails because of an ISRuntimeException.
2	Integration Server suspends document processing and document retrieval for the webMethods messaging trigger temporarily. The webMethods messaging trigger is suspended on this Integration Server only. If the Integration Server is part of a cluster, other servers in the cluster can retrieve and process documents for the webMethods messaging trigger. **Note:**

Step	Description
	The change to the webMethods messaging trigger state is temporary. Document retrieval and document processing will resume for the webMethods messaging trigger if Integration Server restarts, the webMethods messaging trigger is enabled or disabled, or the package containing the webMethods messaging trigger reloads. You can also resume document retrieval and document processing manually using Integration Server Administrator or by invoking the pub.trigger:resumeRetrieval and pub.trigger:resumeProcessing public services.
3	Integration Server rolls back the document to the webMethods messaging trigger document store. This indicates that the required resources are not ready to process the document and makes the document available for processing at a later time. For serial webMethods messaging triggers, it also ensures that the document maintains its position at the top of webMethods messaging trigger queue. **Note:** When the watt.server.dispatcher.messageStore.redeliverOriginalMessage parameter is set to true,Integration Server stores and resubmits the original message after retry failure. If the parameter is set to false, Integration Server stores the message as it is at that point in trigger service execution. If the trigger service modified the message, Integration Server stores the modified message and uses that as input for subsequent trigger service execution. The default value of the watt.server.dispatcher.messageStore.redeliverOriginalMessage parameter is false.
4	Optionally, Integration Server schedules and executes a resource monitoring service. A *resource monitoring service* is a service that you create to determine whether the resources associated with a trigger service are available. A resource monitoring service returns a single output parameter named *isAvailable*.
5	If the resource monitoring service indicates that the resources are available (that is, the value of *isAvailable* is true), Integration Server resumes document retrieval and document processing for the webMethods messaging trigger. If the resource monitoring service indicates that the resources are not available (that is, the value of *isAvailable* is false), Integration Server waits a short time interval (by default, 60 seconds) and then re-executes the resource monitoring service. Integration Server continues executing the resource monitoring service periodically until the service indicates the resources are available. **Tip:** You can change the frequency at which the resource monitoring service executes by modifying the value of the watt.server.trigger.monitoringInterval property.
6	After Integration Server resumes the webMethods messaging trigger, Integration Server passes the document to the webMethods messaging trigger. The webMethods messaging trigger and trigger service process the document just as they would any document in the trigger queue.

Step	Description
	Note: At this point, the retry count is set to 0 (zero).

Configuring Transient Error Handling for a webMethods Messaging Trigger

The transient error handling behavior that you specify for a webMethods messaging trigger determines how Integration Server handles transient errors that occur during trigger service execution. The selected behavior also determines how Integration Server handles transient errors that occur during trigger preprocessing.

For more information about transient error handling for trigger preprocessing, see "Transient Error Handling During Trigger Preprocessing" on page 737.

To configure transient error handling for a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to configure retry behavior.

2. In the Properties view, under **Transient error handling**, select one of the following for **Retry until** property:

Select...	To...
Max attempts reached	Specify that Integration Server retries the trigger service a limited number of times. In the **Max retry attempts** property, enter the maximum number of times Integration Server should attempt to re-execute the trigger service. The default is 0 retries.
Successful	Specify that the Integration Server retries the trigger service until the service executes to completion. **Note:** If a webMethods messaging trigger is configured to retry until successful and a transient error condition is never remedied, a trigger service enters into an infinite retry situation in which it continually re-executes the service at the specified retry interval. Because you cannot disable a webMethods messaging trigger during trigger service execution and you cannot shut down the server during trigger service execution, an infinite retry situation can cause the Integration Server to become unresponsive to a shutdown request. For information about

Select...	To...
	escaping an infinite retry loop, see "About Retrying Trigger Services and Shutdown Requests" on page 722.

3. In the **Retry interval** property, specify the time period the Integration Server waits between retry attempts. The default is 10 seconds.

4. Set the **On retry failure** property to one of the following:

Select...	To...
Throw exception	Indicate that Integration Server throws a service exception when the last allowed retry attempt ends because of an ISRuntimeException. This is the default. For more information about the **Throw exception** option, see "Overview of Throw Exception for Retry Failure" on page 717.
Suspend and retry later	Indicate that Integration Server suspends the webMethods messaging trigger when the last allowed retry attempt ends because of an ISRuntimeException. Integration Server retries the trigger service at a later time. For more information about the **Suspend and retry later** option, see "Overview of Suspend and Retry Later for Retry Failure" on page 718. **Note:** If you want Integration Server to suspend the webMethods messaging trigger and retry it later, you must provide a resource monitoring service that Integration Server can execute to determine when to resume the webMethods messaging trigger. For more information about building a resource monitoring service, see *Publish-Subscribe Developer's Guide*.

5. If you selected **Suspend and retry later**, then in the **Resource monitoring service** property, specify the service that Integration Server should execute to determine the availability of resources associated with the trigger service. Multiple webMethods messaging triggers can use the same resource monitoring service.

6. Click **File > Save**.

Notes:

- webMethods messaging triggers and services can both be configured to retry. When a webMethods messaging trigger invokes a service (that is, the service functions as a trigger service), the Integration Server uses the webMethods messaging trigger retry properties instead of the service retry properties.

- When Integration Server retries a trigger service and the trigger service is configured to generate audit data on error, Integration Server adds an entry to the service log for each failed retry attempt. Each of these entries will have a status of “Retried” and an error message of “Null”. However, if Integration Server makes the maximum retry attempts and the trigger service still fails, the final service log entry for the service will have a status of “Failed” and will display the actual error message. This occurs regardless of which retry failure option the webMethods messaging trigger uses.
- Integration Server generates the following journal log message between retry attempts:

  [ISS.0014.0031D] Service *serviceName* failed with ISRuntimeException. Retry *x* of *y* will begin in *retryInterval* milliseconds.
- If you do not configure service retry for a webMethods messaging trigger, set the **Max retry attempts** property to 0. This can improve the performance of services invoked by the webMethods messaging trigger.
- You can invoke the pub.flow:getRetryCount service within a trigger service to determine the current number of retry attempts made by the Integration Server and the maximum number of retry attempts allowed for the trigger service. For more information about the pub.flow:getRetryCount service, see the *webMethods Integration Server Built-In Services Reference*.

About Retrying Trigger Services and Shutdown Requests

While Integration Server retries a trigger service, Integration Server ignores requests to shut down the server until the trigger service executes successfully or all retry attempts are made. This allows Integration Server to process a document to completion before shutting down.

Sometimes, however, you might want Integration Server to shut down without completing all retries for trigger services. Integration Server provides a server parameter that you can use to indicate that a request to shut down the Integration Server should interrupt the retry process for trigger services. The watt.server.trigger.interruptRetryOnShutdown parameter can be set to one of the following:

Set to...	To...
`false`	Indicate that Integration Server should not interrupt the trigger service retry process to respond to a shutdown request. The Integration Server shuts down only after it makes all the retry attempts or the trigger service executes successfully. This is the default value. **Important:** If watt.server.trigger.interruptRetryOnShutdown is set to “false” and a webMethods messaging trigger is set to retry until successful, a trigger service can enter into an infinite retry situation. If the transient error condition that causes the retry is not resolved, Integration Server continually re-executes the service at the specified retry interval. Because you cannot disable a webMethods messaging trigger during trigger service execution and you cannot shut down the server during

Set to...	To...
	trigger service execution, an infinite retry situation can cause Integration Server to become unresponsive to a shutdown request. To escape an infinite retry situation, set the watt.server.trigger.interruptRetryOnShutdown to “true”. The change takes effect immediately.
`true`	Indicate that Integration Server should interrupt the trigger service retry process if a shutdown request occurs. Specifically, after the shutdown request occurs, Integration Server waits for the current service retry to complete. If the trigger service needs to be retried again (the service ends because of an ISRuntimeException), the Integration Server stops the retry process and shuts down. Upon restart, the transport (the webMethods Broker or, for a local publish, the transient store) redelivers the document to the webMethods messaging trigger for processing. **Note:** If the trigger service retry process is interrupted and the transport redelivers the document to the webMethods messaging trigger, the transport increases the redelivery count for the document. If the webMethods messaging trigger is configured to detect duplicates but does not use a document history database or a document resolver service to perform duplicate detection, Integration Server considers the redelivered document to be “In Doubt” and will not process the document. For more information about duplicate detection and exactly-once processing, see “Exactly-Once Processing for webMethods Messaging Triggers” on page 723.

Note:
When you change the value of the watt.server.trigger.interruptRetryOnShutdown parameter, the change takes effect immediately.

Exactly-Once Processing for webMethods Messaging Triggers

Within Integration Server, exactly-once processing is a facility that ensures one-time processing of a guaranteed document by a webMethods messaging trigger. The webMethods messaging trigger does not process duplicates of the document.

Integration Server provides exactly-once processing for documents received by a webMethods messaging trigger when all of the following are true:

- The document is guaranteed.
- Exactly once properties are configured for the webMethods messaging trigger.

Duplicate Detection Methods for a webMethods Messaging Trigger

Integration Server ensures exactly-once processing by performing duplicate detection and by providing the ability to retry trigger services. Duplicate detection determines whether the current document is a copy of one previously processed by the webMethods messaging trigger. Duplicate documents can be introduced in to the webMethods system when:

- The publishing client publishes the same document more than once.
- During publishing or retrieval of guaranteed documents, the sending resource loses connectivity to the destination resource before receiving a positive acknowledgement for the document. The sending resource will redeliver the document when the connection is restored.

Note:
Exactly-once processing and duplicate detection are performed for guaranteed documents only.

Integration Server uses duplicate detection to determine the document's status. The document status can be one of the following:

- **New.** The document is new and has not been processed by the webMethods messaging trigger.
- **Duplicate.** The document is a copy of one already processed the webMethods messaging trigger.
- **In Doubt.** Integration Server cannot determine the status of the document. The webMethods messaging trigger may or may not have processed the document before.

To resolve the document status, Integration Server evaluates, in order, one or more of the following:

- **Redelivery count** indicates how many times the transport has redelivered the document to the webMethods messaging trigger.
- **Document history database** maintains a record of all guaranteed documents processed by webMethods messaging triggers for which exactly-once processing is configured.
- **Document resolver service** is a service created by a user to determine the document status. The document resolver service can be used instead of or in addition to the document history database.

The steps that Integration Server performs to determine a document's status depend on the exactly-once properties configured for the subscribing trigger. For more information about configuring exactly-once properties, see "Configuring Exactly-Once Processing for a webMethods Messaging Trigger " on page 725.

Note:
For detailed information about exactly-once processing for webMethods messaging triggers, see *Publish-Subscribe Developer's Guide*.

Configuring Exactly-Once Processing for a webMethods Messaging Trigger

Configure exactly-once processing for a webMethods messaging trigger when you want the webMethods messaging trigger to process guaranteed documents once and only once. If it is acceptable for a trigger service to process duplicates of a document, you should not configure exactly-once processing for the webMethods messaging trigger.

Keep the following points in mind when configuring exactly-once processing:

- Integration Server can perform exactly-once processing for guaranteed documents only.
- You do not need to configure all three methods of duplicate detection. However, if you want to ensure exactly-once processing, you must use a document history database or implement a custom solution using the document resolver service.

  A document history database offers a simpler approach than building a custom solution and will typically catch all duplicate messages. There may be exceptions depending on your implementation. For more information about these exceptions, see *Publish-Subscribe Developer's Guide*. To minimize these exceptions, it is recommended that you use a history database and a document resolver service.
- If Integration Server connects to an 6.0 or 6.0.1 version of the webMethods Broker, you must use a document history database and/or a document resolver service to perform duplicate detection. Earlier versions of the webMethods Broker do not maintain a redelivery count. Integration Server will assign documents received from these webMethods Brokers a redelivery count of -1. If you do not enable another method of duplicate detection, Integration Server assigns the document a New status and executes the trigger service.
- Stand-alone Integration Servers cannot share a document history database. Only a cluster or a non-clustered group of Integration Servers can share a document history database.
- Make sure the duplicate detection window set by the **History time to live** property is long enough to catch duplicate documents but does not cause the document history database to consume too many server resources. If external applications reliably publish documents once, you might use a smaller duplicate detection window. If the external applications are prone to publishing duplicate documents, consider setting a longer duplicate detection window.
- If you intend to use a document history database as part of duplicate detection, you must first install the document history database component and associate it with a JDBC connection pool. For instructions, see *Installing Software AG Products*.

To configure exactly-once processing for a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to configure exactly-once processing.
2. In the Properties view, under **Exactly Once**, set the **Detect duplicates** property to **True**.

3. To use a document history database as part of duplicate detection, do the following:

   a. Set the **Use history** property to **True**.

   b. In the **History time to live** property, specify how long the document history database maintains an entry for a message processed by this webMethods messaging trigger. This value determines the length of the duplicate detection window.

4. To use a service that you create to resolve the status of In Doubt messages, specify that service in the **Document resolver service** property.

5. Click **File > Save**.

Disabling Exactly-Once Processing for a webMethods Messaging Trigger

If you later determine that exactly-once processing is not necessary for a webMethods messaging trigger, you can disable it. When you disable exactly-once processing, the Integration Server provides at-least-once processing for all guaranteed documents received by the webMethods messaging trigger.

To disable exactly-once processing for a webMethods messaging trigger

1. In the Package Navigator view of Designer, open the webMethods messaging trigger for which you want to configure exactly-once processing.

2. In the Properties view, under **Exactly Once**, set the **Detect duplicates** property to **False**.

   Designer disables the remaining exactly-once properties.

3. Click **File > Save**.

Modifying a webMethods Messaging Trigger

After you create a webMethods messaging trigger, you can modify it by changing or renaming the condition, specifying different publishable document types, specifying different trigger services, or changing webMethods messaging trigger properties. To modify a webMethods messaging trigger, you need to lock the webMethods messaging trigger and have write access to the webMethods messaging trigger.

If your integration solution includes a messaging provider (webMethods Broker or Universal Messaging), the messaging provider needs to be available when editing a webMethods messaging trigger. Editing a webMethods messaging trigger when the messaging provider is unavailable can cause the webMethods messaging trigger to become out of sync with the associated object on the provider. Do not edit any of the following webMethods messaging trigger components when the messaging provider used by the publishable document types in the trigger are not available.

- Any publishable document types specified in the webMethods messaging trigger. That is, do not change the subscriptions established by the webMethods messaging trigger.
- Any filters specified in the webMethods messaging trigger.
- The webMethods messaging trigger state (enabled or disabled).
- The document processing mode (serial or concurrent processing).
- Priority messaging state (enabled or disabled). This applies to webMethods messaging triggers that receive documents from the webMethods Broker only.

If you edit any of these webMethods messaging trigger components when messaging provider is unavailable and save the changes, the webMethods messaging trigger will become out of sync with its associated object on the messaging provider. You will need to synchronize the webMethods messaging trigger with its associated provider object when the messaging provider becomes available. To synchronize, use Designer to disable the webMethods messaging trigger, re-enable the webMethods messaging trigger, and save. This effectively recreates the object on the messaging provider that is associated with the webMethods messaging trigger. Alternatively, you can disable and then enable the messaging connection alias used by the trigger. However, this restarts all the triggers that use the messaging connection alias and may consume more time and resources.

Note:
If you changed the message processing mode for a webMethods messaging trigger that uses a Universal Messaging connection alias with a shared client prefix, you might need to use Universal Messaging Enterprise Manager to delete and recreate the named object. For more information, see "Synchronizing the webMethods Messaging Trigger and Named Object on Universal Messaging " on page 713.

Modifying a webMethods Messaging Trigger in a Cluster or Non-Clustered Group

Once you set up a cluster or a non-clustered group of Integration Servers, avoid editing any of the webMethods messaging triggers in the cluster or non-clustered group.

Important:
Modifying triggers on an Integration Server in a cluster or in a non-clustered group of servers can cause the triggers to be out of sync with the other servers in the cluster or non-clustered group. It can also create inconsistencies with the associated object on the messaging provider.

In a clustered environment, you can modify selected webMethods messaging trigger properties (capacity, refill level, maximum and execution threads) using the Integration Server Administrator. For more information about editing webMethods messaging trigger properties using the Integration Server Administrator, see *webMethods Integration Server Administrator's Guide*.

Deleting webMethods Messaging Triggers

Keep the following points in mind when deleting a webMethods messaging trigger:

- When you delete a webMethods messaging trigger, Integration Server deletes the queue for the webMethods messaging trigger on Integration Server.
- If the messaging connection alias used by the publishable document type to which the trigger subscribes does not share a client prefix, deleting the webMethods messaging trigger causes the messaging provider to delete the associated provider object (webMethods Broker client queue or Universal Messaging named object on a channel).
- If the messaging connection alias used by the publishable document type to which the trigger subscribes shares a client prefix, deleting the webMethods messaging trigger does not cause the messaging provider to delete the associated provider object (webMethods Broker client queue or Universal Messaging named object on a channel).
- To delete a webMethods messaging trigger, you must lock it and have write access to it.
- You can also use the pub.trigger:deleteTrigger service to delete a webMethods messaging trigger. For more information about this service, see the *webMethods Integration Server Built-In Services Reference*.

To delete a webMethods messaging trigger

1. In the Package Navigator view of Designer, select the webMethods messaging trigger that you want to delete.
2. Click **Edit > Delete**.
3. In the Delete Confirmation dialog box, click **OK**.

Deleting webMethods Messaging Triggers in a Cluster or Non-Clustered Group

When a webMethods messaging trigger exists on multiple Integration Servers in a cluster or non-clustered group of Integration Servers, the subscriptions created by the webMethods messaging trigger remain active even if you delete the webMethods messaging trigger from one of the Integration Servers. When deleting webMethods messaging triggers from the servers in a cluster or non-clustered group, the associated provider object remains connected to the cluster or non-clustered group until you delete the webMethods messaging trigger on all of the servers. If you do not delete the webMethods messaging trigger on all of the servers, the provider object for the webMethods messaging trigger remains connected and the messaging provider continues to enqueue documents for the webMethods messaging trigger.

To delete a webMethods messaging trigger from a cluster or non-clustered group of Integration Servers, delete the webMethods messaging trigger from each Integration Server in the cluster, and then manually delete the provider object associated with webMethods messaging trigger from the messaging provider.

Note:

In addition to the term “non-clustered group,” the terms “stateless cluster” and “external cluster” are sometimes used to describe the situation in which a group of Integration Servers function in a manner similar to a cluster but are not part of a configured cluster.

Running a webMethods Messaging Trigger with a Launch Configuration

In Designer, you can run a webMethods messaging trigger to verify that the subscription, filters, and trigger service work as expected. Designer requires launch configurations to run webMethods messaging triggers. However, if a webMethods messaging trigger does not have an associated launch configuration and you bypass the Run Configurations dialog boxes when running the webMethods messaging trigger, Designer creates a launch configuration on the fly and saves it in your workspace. You can use this configuration from one session to the next. In fact, Designer reuses this configuration every time you run the webMethods messaging trigger without creating another launch configuration.

By default, Designer saves launch configurations locally in an unexposed location in your workspace. However, you might want to share launch configurations with other developers. You can specify that Designer save a launch configuration to a shared file within your workspace; this location will be exposed. On the **Common** tab in the Run Configurations dialog box, select the **Shared file** option and provide a workspace location in which to save the file.

In a launch configuration for a webMethods messaging trigger, you specify:

- The condition that you want Designer to test. Each launch configuration can specify only one condition in the webMethods messaging trigger.
- The document type whose subscription you want to test. For an Any (OR) or Only one (XOR) join condition, you specify the document type for which you want to supply input.
- Input data that Designer uses to build a document. Designer evaluates the filter using the data in the document and provides the document as input to the trigger service.

You can create multiple launch configurations for each webMethods messaging trigger.

Creating a Launch Configuration for a webMethods Messaging Trigger

Use the following procedure to create a launch configuration for running a webMethods messaging trigger.

To create a launch configuration for running a webMethods messaging trigger

1. In the Service Development perspective, select **Run > Run Configurations**
2. In the **Run Configuration** dialog box, select **webMethods Messaging Trigger** and click to add a new launch configuration.

3. In the **Name** field, specify a name for the launch configuration.

4. On the **webMethods Messaging Trigger** tab, in the **Integration Server** list, select the Integration Server on which the webMethods messaging trigger for which you are creating a launch configuration resides.

5. In the **webMethods Messaging Trigger** field click **Browse** to navigate to and select the trigger.

6. On the **Input** tab, in the **Condition** list, select the condition that you want to test using the launch configuration.

7. If the condition is a join condition with an Any (OR) or Only one (XOR) join, do the following:

   a. Next to **Document Type**, click **Select**.

   b. In the Select a Document Type dialog box, select the document type for which you want to provide input data in this launch configuration.

      - For an Any (OR) join, select one document type.
      - For an Only one (XOR) join, select the document or document types that you want to use to test the join condition.

   c. Click **OK**.

8. On the **Input** tab, select the tab with the name of the IS document type for which you want to provide input data.

   If the selected condition uses an All (AND) join, Designer displays one tab for each document type in the join condition. If the condition is an Only one (XOR) join and you selected multiple document types for which to supply input data, Designer displays one tab for each selected document type.

   a. Select or clear the **Include empty values for String Types** check box to indicate how to handle variables that have no value.

      - If you want to use an empty String (i.e., a String with a zero-length), select the **Include empty values for String Types** check box. Also note that Document Lists that have defined elements will be part of the input, but they will be empty.
      - If you want to use a null value for the empty Strings, clear the check box. String-type variables will not included in the input document.

      Note:
      The setting applies to all String-type variables in the root document of the input signature. The setting does not apply to String-type variables within Document Lists. You define how you want to handle String-type variables within Document Lists separately when you assign values to Document Lists variables.

b. Specify the values to save with the launch configuration for the webMethods messaging trigger by doing one of the following:

   - Type the input value for each field in the document type.
   - To load the input values from a file, click **Load** to locate and select the file containing the input values. If Designer cannot parse the input data, it displays an error message indicating why it cannot load the data.

   Designer validates the provided input values. If provided values do not match the input parameter data type, Designer displays a message to that effect. You cannot use the launch configuration for the webMethods messaging trigger if the provided input does not match the defined data type.

c. If you want Designer to give the user executing the launch configuration the option of providing different input values than those saved with the launch configuration, select the **Prompt for data at launch** check box. If you clear this check box, Designer passes the webMethods messaging trigger the same set of data every time the launch configuration executes.

9. Repeat the preceding step for each IS document type displayed on the **Input** tab.

10. If you want to save the input values that you have entered, click **Save**.

11. Click **Apply**.

12. If you want to execute the launch configuration, click **Run**. Otherwise, click **Close**.

Running a webMethods Messaging Trigger

Keep the following points in mind when running a webMethods messaging trigger:

- When you run a webMethods messaging trigger, you can select the launch configuration that Designer uses to run the webMethods messaging trigger. If a launch configuration does not exist for a webMethods messaging trigger, Designer creates a launch configuration and immediately prompts you for input values and then runs the webMethods messaging trigger. Designer saves the launch configuration in your workspace.
- When you run a webMethods messaging trigger, you can only test one condition at a time.
- When you run a webMethods messaging trigger with a join condition Designer treats the activation IDs for the documents as identical. Designer ignores the value of the *activation* field in the document envelope.
- When you run a webMethods messaging trigger by running a launch configuration, the webMethods messaging trigger is tested locally. That is, a document is not routed through the messaging provider.

  Note:

To test a webMethods messaging trigger by publishing a document to the messaging provider, test a publishable document type. You test a publishable document type by creating and running a launch configuration for the publishable document type.

- Designer displays results for running the webMethods messaging trigger in the Results view.

To run a webMethods messaging trigger

1. In Package Navigator view of the Service Development perspective, select the webMethods messaging trigger you want to run.

2. Select **Run > Run As > webMethods Messaging Trigger**

3. If multiple launch configurations exist for the service, use the **Select Launch Configuration** dialog box to select the launch configuration that you want Designer to use to run the webMethods messaging trigger.

4. If the launch configuration is set up to prompt the user for input values or there is no launch configuration, in the **Enter Input for *triggerName*** dialog box, in the **Condition** list, select the condition that you want to test using the launch configuration.

5. If the condition is a join condition with an Any (OR) or Only one (XOR) join, do the following:

   a. Next to **Document type**, click **Select**.

   b. In the Select a Document Type dialog box, select the document type for which you want to provide input data.

      - For an Any (OR) join, select one document type.
      - For an Only one (XOR) join, select one or more document types to use to test the join condition. Note that at run time, the trigger service processes only one of the documents. The trigger discards the other document.

   c. Click **OK**.

6. In the **Enter Input for *triggerName*** dialog box, select the tab with the name of the IS document type for which you want to provide input data.

7. Select or clear the **Include empty values for String Types** check box to indicate how to handle variables that have no value.

   - If you want to use an empty String (i.e., a String with a zero-length), select the **Include empty values for String Types** check box. Also note that Document Lists that have defined elements will be part of the input, but they will be empty.
   - If you want to use a null value for the empty Strings, clear the check box. String-type variables will not included in the input document.

Note:
The setting applies to all String-type variables in the root document of the input signature. The setting does not apply to String-type variables within Document Lists. You define how you want to handle String-type variables within Document Lists separately when you assign values to Document Lists variables. For more information, see *webMethods Service Development Help*.

8. Specify the values to save with the launch configuration for the webMethods messaging trigger by doing one of the following:

   - Type the input value for each field in the document type.
   - To load the input values from a file, click **Load** to locate and select the file containing the input values. If Designer cannot parse the input data, it displays an error message indicating why it cannot load the data.

   Note:
   If you type in input values, Designer discards the values you specified after the run. If you want to save input values, create a launch configuration. For instructions, see "Running a webMethods Messaging Trigger " on page 731.

9. Click **OK**.

   Designer runs the trigger and displays the results in the Results view.

Testing Join Conditions

While running a launch configuration for a webMethods messaging trigger provides verification of filters and the trigger service execution, it does not test all aspects of a join condition. For example, running a webMethods messaging trigger does not test the join expiration. In addition to running a launch configuration to test a join condition, consider testing the join condition in the following ways:

- Publish documents from Designer using a launch configuration.

  You can publish documents by creating and running a launch configuration for a publishable document type.

  To test a join condition by publishing documents via a launch configuration, you must use the same activation ID for all the documents specified in the join. If you re-use an activation ID from one test to the next, make sure that the documents sent in the first test are processed before starting the next test.

- Create a service that publishes the documents.

  You can also test join processing for a join condition by creating a flow service that invokes a publish service for each of the document types specified in the join condition. Integration Server automatically assigns an activation ID and uses that activation ID for all the documents published in the same service.

During trigger processing and join processing, Integration Server writes messages to the journal log. You can use the contents of the journal log to test and debug the join conditions in the webMethods messaging trigger.

Debugging a webMethods Messaging Trigger

To debug and test a webMethods messaging trigger, you can:

- Run the webMethods messaging trigger using a launch configuration. You can use the launch configuration to verify that the subscription, filters, and trigger service work as expected.
- Publish a document to which the webMethods messaging trigger subscribes. You can do this by creating a launch configuration for a publishable document type and then running the launch configuration. Alternatively, you can create a service that uses one of the pub.publish* services to publish a document to which the trigger subscribes and then run the service to publish the document.
- Instruct Integration Server to produce an extra level of verbose logging. You can enable debug trace logging for all webMethods messaging triggers or an individual webMethods messaging trigger.

  Note: Integration Server generates additional logging for triggers that receive messages from Universal Messaging or through Digital Event Services only.

Enabling Trace Logging for All webMethods Messaging Triggers

You can configure Integration Server to generate additional debug and trace logging for all the webMethods messaging triggers that receive messages from Universal Messaging or through Digital Event Services.

Note:
For the increased logging to appear in the server log, you must set the logging level for server facility 0153 Dispatcher (Universal Messaging) to Trace.

To enable debug trace logging for all webMethods messaging triggers

1. Open Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the Navigation panel, click **Extended**.
3. Click **Edit Extended Settings**.
4. Under **Extended Settings**, type the following:

   ```
   watt.server.messaging.debugTrace=true
   ```

5. Click **Save Changes**.

6. Disable and then enable the messaging connection aliases used by the webMethods messaging triggers.

Enabling Trace Logging for a Specific webMethods Messaging Trigger

You can configure Integration Server to generate additional debug and trace logging for a specific webMethods messaging trigger that receives messages from Universal Messaging or through Digital Event Services.

Note:
For the increased logging to appear in the server log, you must set the logging level for server facility 0153 Dispatcher (Universal Messaging) to Trace.

To enable debug trace logging for a specific webMethods messaging trigger

1. Open Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the Navigation panel, click **Extended**.
3. Click **Edit Extended Settings**.
4. Under **Extended Settings**, type the following:

   ```
   watt.server.messaging.debugTrace.triggerName=true
   ```

   Where *triggerName* is the fully qualified name of the trigger in the format *folder*.*subfolder*:*triggerName*.
5. Click **Save Changes**.
6. Disable and then enable the trigger.

34 Transient Error Handling During Trigger Preprocessing

- Server and Trigger Properties that Affect Transient Error Handling During Trigger Preprocessing 738
- Overview of Transient Error Handling During Trigger Preprocessing 739

Trigger preprocessing encompasses the time from when a trigger first receives a message (document) from its local queue on Integration Server to the time Integration Server invokes the trigger service. Transient errors can occur during this time. A transient error is an error that arises from a temporary condition that might be resolved or corrected quickly, such as the unavailability of a resource due to network issues or failure to connect to a database. For example, if a document history database is used for exactly-once processing, the unavailability of the database may cause a transient error. Because the condition that caused the trigger preprocessing to fail is temporary, the trigger preprocessing might complete successfully if Integration Server waits and then re-attempts trigger preprocessing. To allow the preprocessing to complete successfully, Integration Server provides some properties and settings for transient error handling.

Server and Trigger Properties that Affect Transient Error Handling During Trigger Preprocessing

Integration Server and Designer provide properties that you can use to configure how Integration Server handles transient errors that occur during the preprocessing phase of trigger execution.

- The watt.server.trigger.preprocess.suspendAndRetryOnError server configuration property. This property determines if Integration Server suspends a trigger if an error occurs during trigger preprocessing. This server configuration parameter acts as a global on/off switch. When set to true, Integration Server suspends any trigger that experiences an error during preprocessing. When set to false, Integration Server uses the individual trigger properties to determine whether or not to suspend the trigger.
- The watt.server.trigger.preprocess.monitorDatabaseOnConnectionException server configuration property. This property determines how Integration Server handles a ConnectionException that causes a transient error. A ConnectionException occurs when the document history database is not enabled or is configured incorrectly.
- The **On Retry Failure** trigger property for webMethods messaging triggers and non-transacted JMS triggers. When set to **Suspend and retry later**, Integration Server suspends a trigger that encounters a transient error during trigger preprocessing.

  Note:
  The **On Retry Failure** trigger property also determines how Integration Server handles retry failure for a trigger service.

- The **On Transaction Rollback** property for a transacted JMS trigger. When set to Suspend and recover, Integration Server suspends a transected JMS trigger that encounters a transient error during trigger preprocessing.

  Note:
  The **On Transaction Rollback** property also determines how Integration Server handles a transaction rollback caused by a transient error that occurs during trigger execution.

For a detailed explanation about how Integration Server uses these property settings when a transient error occurs during trigger preprocessing, see "Overview of Transient Error Handling During Trigger Preprocessing" on page 739.

Overview of Transient Error Handling During Trigger Preprocessing

Following is an overview of how Integration Server performs transient error handling for an ISRuntimeException that occurs during trigger preprocessing. Typically, transient errors that occur during preprocessing occur during exactly-once processing. For example, the document history database might not be available of the document resolver service fails because of an ISRuntimeException.

Step	Description
1	A transient error, specifically an ISRuntimeException, occurs during the preprocessing phase of trigger execution.
2	Integration Server checks the values of watt.server.trigger.preprocess.suspendAndRetryOnError server configuration property and the **On Retry Failure** trigger property. If this is a transacted JMS trigger, Integration Serverchecks the value of the **On Transaction Rollback** property instead of the **On Retry Failure** property. If one of the following is true, Integration Server suspends the trigger, rolls the message back to the messaging provider, and proceeds as described in step **3**: ■ watt.server.trigger.preprocess.suspendAndRetryOnError is set to true. ■ **On Retry Failure** property is set to **Suspend and retry later** or **On Transaction Rollback** property is set to **Suspend and recover**. If none of the above are true, then Integration Server does not suspend the trigger if a transient error occurs during trigger preprocessing. Instead, Integration Server does one of the following: ■ If the trigger specifies a document resolver service, Integration Server executes the document resolver service to determine the status of the document. If the document resolver service ends because of an ISRuntimeException, Integration Server assigns the document a status of In Doubt, acknowledges the document, and uses the audit subsystem to log the document. ■ If the trigger does not specify a document resolver service, Integration Server assigns the document a status of In Doubt. Integration Server throws an exception, acknowledges the document to the messaging provider, and uses the audit subsystem to log the document. This may result in message loss. **Note:** If the trigger is a webMethods messaging trigger, Integration Server uses the audit subsystem to log the document. You can use webMethods Monitor to resubmit the document.
3	Integration Server does one of the following once the trigger is suspended:

Step	Description
	■ If the transient error (ISRuntimeException) is caused by a SQLException (which indicates that an error occurred while reading to or writing from the database), Integration Server suspends the trigger and schedules a system task that executes an internal service that monitors the connection to the document history database. Integration Server resumes the trigger and re-executes it when the internal service indicates that the connection to the document history database is available. ■ If the transient error (ISRuntimeException) is caused by a ConnectionException (which indicates that document history database is not enabled or is not properly configured), and the watt.server.trigger.preprocess.monitorDatabaseOnConnectionException property is set to true, Integration Server schedules a system task that executes an internal service that monitors the connection to the document history database. Integration Server resumes the trigger and re-executes it when the internal service indicates that the connection to the document history database is available. ■ If the transient error (ISRuntimeException) is caused by a ConnectionException and the watt.server.trigger.preprocess.monitorDatabaseOnConnectionException property is set to false, Integration Server does not schedule a system task to check for the database's availability and will not resume the trigger automatically. You must manually resume the trigger after configuring the document history database properly. ■ If the transient error (ISRuntimeException) is caused by some other type of exception, Integration Server suspends the trigger and schedules a system task to execute the trigger's resource monitoring service (if one is specified). When the resource monitoring service indicates that the resources used by the trigger are available, Integration Server resumes the trigger and again receives the message from the messaging provider. If a resource monitoring service is not specified, you will need to resume the trigger manually (via Integration Server Administrator or the pub.trigger* services).

35 How Triggers Correspond to Durable Subscribers on Universal Messaging

When Universal Messaging is used as the provider for webMethods messaging or JMS, triggers on Integration Server may map to a durable on Universal Messaging, specifically:

- When Universal Messaging is used as the webMethods messaging provider, the webMethods messaging triggers map to durable subscribers on Universal Messaging. Integration Server configures this automatically for webMethods messaging triggers.
- When Universal Messaging is the JMS provider, the JMS triggers that receive messages from a topic with a JMS durable subscription do use a durable on Universal Messaging. The JMS triggers receive messages from a channel on Universal Messaging with a durable, either shared or serial.

  For a JMS trigger that uses a durable subscriber to retrieve messages from a topic on Universal Messaging, the type of durable is set on the connection factory. The type of durable is not set on or by Integration Server. You must use Universal Messaging Enterprise Manager to set the durable on the connection factory. Software AG recommends that you set the type of durable based on the processing mode for the JMS trigger.

Note:
JMS triggers that receive messages from a queue do *not* use a durable on Universal Messaging. The triggers receive messages from a queue on Universal Messaging. JMS triggers that receive messages from a topic without a durable subscription do *not* use a durable on Universal Messaging either. The triggers receive messages from a channel on Universal Messaging

Knowing which durable corresponds to a particular type of trigger can help you locate information in the Universal Messaging documentation about the durable. This may provide a more complete picture about the behavior of the trigger and its durable as well as make it easier to troubleshoot issues that may occur.

The following table identifies the durable type used with a specific combination of the trigger type and message processing mode.

Trigger Type	Processing Mode	Durable Type
webMethods messaging trigger	Serial	Priority **Note:**

Trigger Type	Processing Mode	Durable Type
		When a webMethods messaging trigger created and running on Integration Server 10.1 connects to Universal Messaging 10.1 or 10.3, the serial trigger corresponds to a priority durable. However, when a serial trigger created and running on Integration Server 10.1 connects to Universal Messaging 10.5 or higher, a serial durable will be created for the trigger. Universal Messaging 10.5 and higher supports serial durables but not priority durables.
webMethods messaging trigger	Concurrent	Shared
JMS trigger that receives messages from a topic for which there is a durable subscription	Serial	Serial **Note:** This is the recommended durable type for the serial processing mode. You must set this on the connection factory.
JMS trigger that receives messages from a topic for which there is a durable subscription	Concurrent	Shared **Note:** This is the recommended durable type for the concurrent processing mode. You must set this on the connection factory.

Note:
For a JMS trigger that uses a durable subscriber to retrieve messages from a topic on Universal Messaging, the type of durable is controlled by the connection factory. The type of durable is not set automatically Integration Server. Software AG recommends that you set the durable type on a connection factory to match the processing mode identified in the above table.

36 Working with Web Services

- What Are Web Service Descriptors? 745
- About Provider Web Service Descriptors 746
- About Consumer Web Service Descriptors 760
- About Refreshing a Web Service Descriptor 771
- Viewing the WSDL Document for a Web Service Descriptor 781
- WS-I Compliance for Web Service Descriptors 782
- Changing the Target Namespace for a Web Service Descriptor 784
- Viewing the Namespaces Used within a WSDL Document 784
- Enabling MTOM/XOP Support for a Web Service Descriptor 784
- Adding SOAP Headers to the Pipeline 785
- Validating SOAP Response 786
- Validating Schemas Associated with a Web Service Descriptor 787
- Omitting xsd:any from the WSDL Document 789
- Working with Binders 790
- Working with Operations 799
- Adding Headers to an Operation 805
- About SOAP Fault Processing 807
- Viewing Document Types for a Header or Fault Element 813
- Working with Handlers 813

- Working with Policies 816
- About Pre-8.2 Compatibility Mode 818

Web services are building blocks for creating open, distributed systems. A web service is a collection of functions that are packaged as a single unit and published to a network for use by other software programs. For example, you could create a web service that checks a customer's credit or tracks delivery of a package. If you want to provide higher-level functionality, such as a complete order management system, you could create a web service that maps to many different IS flow services, each performing a separate order management function.

Designer uses web service descriptors to encapsulate information about web services and uses web service connectors to invoke web services.

Note:
Information about web services is located in *webMethods Service Development Help*, *Web Services Developer's Guide*, and *webMethods Integration Server Administrator's Guide*.

- *webMethods Service Development Help* includes this "Working with Web Services" on page 743 topic which provides procedures for using Designer to create web service descriptors, adding operations, binders, handlers, and policies to a web service descriptor; and setting web service descriptor properties.
- *Web Services Developer's Guide* contains information such as how Integration Server processes web services, how a SOAP fault is represented in the pipeline, steps to configure MTOM streaming when sending and receiving SOAP messages using web services, and how to secure web services with WS-Security and WSSecurityPolicy. For completeness, *Web Services Developer's Guide* also contains the "Working with Web Services" on page 743 topic that appears in *webMethods Service Development Help*.
- *webMethods Integration Server Administrator's Guide* contains information about creating web service endpoint alias and configuring Integration Server to use web services reliable messaging.

What Are Web Service Descriptors?

A *web service descriptor (WSD)* is an element on Integration Server that defines a web service in IS terms. The WSD encapsulates all the information required by the provider or the consumer (requester) of a web service. The WSD contains the message formats, data types, transport protocols, and transport serialization formats that should be used between the consumer (requester) and the provider of the web service. It also specifies one or more network locations at which a web service can be invoked. In essence, the WSD represents an agreement governing the mechanics of interacting with that service.

- A *provider web service descriptor* defines a web service that is hosted on the Integration Server, that is, a service "provided" to external users. A provider web service descriptor will expose one or more IS services as *operations*, which can be published to a registry as a single web service. External users can access the web service through the registry and invoke the IS services remotely.

- A *consumer web service descriptor* defines an external web service, allowing Integration Server to create a *web service connector (WSC)* for each operation in the web service. The web service connector(s) can be used in Designer just like any other IS flow service; when a connector is invoked it calls a specific operation of a web service. In version 9.0 and later, Integration Server also creates a response service for each operation in the web service. Response services are flow services to which you can add custom logic to process asynchronous SOAP responses.

About Provider Web Service Descriptors

You can turn any service in any Integration Server package into a web service by using the IS service as an operation in a provider web service descriptor. Integration Server provides an environment for executing services efficiently and securely. It receives and decodes requests from clients, calls the requested services, and encodes and returns the output to the clients.

A *provider web service descriptor* (WSD) is created from one or more IS services or from a single WSDL document, and is designed to allow the IS services to be invoked as web services over a network. The provider web service descriptor contains all the data required to create a WSDL document for the IS web service, as well as the data needed at run time to process the request and response.

You can create a provider web service descriptor from a service that exists on Integration Server or from a WSDL document.

- A *service first provider web service descriptor* refers to provider web service descriptors created from an existing service on Integration Server. In this case, you specify the protocol, binding style/use, and host server when creating the WSD. The IS service becomes an operation in the provider web service descriptor. Integration Server uses the existing service signature as the input and output messages for the operation. You can add operations and bindings to a service first provider web service descriptor.
- A *WSDL first provider web service descriptor* refers to a provider web service descriptor created from an existing WSDL document, from a service asset in CentraSite, or from a web service acquired from a UDDI registry. In this case, Designer uses the message and operation definitions from the WSDL to generate a "placeholder" flow service for each operation encountered in the WSDL, along with IS document types defining the input and output signature of the generated flow services. You can then implement any required logic within the placeholder flow service. Note that you cannot add operations or bindings to a WSDL first provider WSD.

The provider web service descriptor can be published to a UDDI registry (or other publicly accessible server) as a web service, which can be invoked remotely by an external user. A web service provider can also distribute WSDL files directly to consumers.

Service Signature Requirements for Service First Provider Web Service Descriptors

When you create a service first provider web service descriptor, you select one or more services to use as operations. The service signature becomes the input and output messages for the operations in the WSDL document. However, Integration Server allows constructs within service signatures that cannot be represented in certain web service style/use combinations.

When adding a service to or creating a service first provider web service descriptor, Integration Server verifies that the service signature can be represented in the style/use specified for the web service descriptor. If a service signature does not meet the style/use signature requirements, Integration Server will not add the service as an operation. Or, in the case of creating a service first provider WSD, Integration Server will not create the WSD.

Following, is a list of service signature restrictions and requirements for each style/use. Note that this list may not be exhaustive.

Signature Restrictions for Document/Literal
*body fields are not allowed at the top level
@attribute fields (fields starting with the "@" symbol) are not allowed at the top level
String table fields are not allowed

Signature Restrictions for RPC/Encoded
* body fields are not allowed
@attribute fields are not allowed (fields starting with the "@" symbol)
Top-level fields cannot be namespace qualified
Top-level field names cannot be in the format *prefix*:*localName*

Signature Restrictions for RPC/Literal
*body fields are not allowed at the top level
@attribute fields (fields starting with the "@" symbol) are not allowed at the top level
String table fields are not allowed
List fields (String List, Document List, Document Reference List, and Object List) are not allowed at the top level
Duplicate field names (identically named fields) are not allowed at the top level
Top-level fields cannot be namespace qualified
Top-level field names cannot be in the format *prefix*:*localName*

Using XML Namespaces with Prefixes with Fields in Service Signatures

You can associate the name of an Integration Server field (such as an IS document variable) with an XML namespace. When you do this, the local name is the name of the field and the XML namespace name is the URI that identifies the namespace. You can also include a prefix as part of the name.

Assign XML namespaces and prefixes to Integration Server fields as follows:

- To assign an XML namespace to an Integration Server field, complete the **XML Namespace** property in the **General** category of the field's Properties view.
- To assign a prefix to an Integration Server field, precede the field name with the prefix followed by a colon (for example, *prefix*:*variableName*).

Note:
The style/use combinations RPC/Literal and RPC/Encoded prohibit top-level field from being namespace qualified.

Handling Incomplete Service Signatures Using Wrapper Services

When you use a service as an operation in a web service descriptor, the service signature must accurately and completely reflect the expected service input and output.

If the signature is not accurate or complete, the WSDL document created for the web service descriptor will contain incorrect signature information. Clients generated from the WSDL document may not execute as expected.

However, sometimes it may not be possible to make the service signature complete before using it in a web service descriptor or you may not want to alter the service signature. In these situations, you can expose the service as a web service by creating a wrapper service. The wrapper service needs to declare the complete service signature and invoke the service that you want to expose as a web service. You can then use the wrapper service as an operation in a provider web service descriptor.

For example, suppose that you want to expose an XSLT service as a web service on one Integration Server and invoke it from another. However, the XSLT source contains an optional run-time property that is added to the pipeline at run time. This optional property is not reflected in the input signature of the XSLT service. If you added the XSLT service to a provider web service descriptor, the resulting WSDL document would not list the property as part of the input message. Consequently, a consumer web service descriptor and a web service connector created from the WSDL document would not account for the property and invocation will fail.

To successfully use the XSLT service as a web service, you can do the following:

1. Create a wrapper flow service that:
   - Defines all of the input parameters of the XSLT service in its input signature.
   - Defines the run-time property of the XSLT source in its input signature.
   - Invokes the XSLT service.
2. On the Integration Server that hosts the wrapper flow service and the XSLT service, create a provider web service descriptor from the wrapper flow service.

On the Integration Server from which you will invoke the web service, create a consumer web service descriptor from the WSDL of the provider web service descriptor. The web service connector that corresponds to the operation for the XSLT service will display the complete input signature.

Creating a Service First Provider Web Service Descriptor

Keep the following points in mind when creating a service first provider web service descriptor:

- You must have Write access to the folder in which you want to store the provider web service descriptor.

- The style and use selected for a provider web service descriptor determines what types of fields and field names are allowed in the service signature. Designer will not create a provider web service descriptor if the signature of the service does not meet the requirements of the selected binding style/use. For more information, see "Service Signature Requirements for Service First Provider Web Service Descriptors" on page 746.
- Depending on the use and style that you specify, you may have to either rename certain fields in the IS service or assign an XML namespace to them.
- You must have at least one web service endpoint alias that specifies the JMS transport before you can create a provider web service descriptor with a JMS binder. For more information about creating a web service endpoint alias, see *webMethods Integration Server Administrator's Guide*.
- When using an adapter service to create a provider web service descriptor, if the service returns values in the pipeline that do not match the output signature, you must change those variable properties to optional fields (where applicable), or else wrap the service in a flow to add or drop variables to match the output signature.
- Web service descriptors that are not running in compatibility mode can stream MTOM attachments for both inbound and outbound SOAP messages. To stream MTOM attachments, the object that represents the field to be streamed should be of type com.wm.util.XOPObject Java class.
- You can quickly create a service first provider web service descriptor by right-clicking the service, selecting **Generate Provider WSD**. Enter a name for the web service descriptor in the Provide a Name dialog box and click **OK**. Designer automatically creates a provider web service descriptor in the same folder as the selected IS service, using all the default options.
- Integration Server generates invalid WSDL for Axis and .Net clients if the provider web service descriptor contains a C service for that takes a document specification as input. Axis and .Net clients cannot handle the resulting Java stub classes and throw an error. Do not use a C service with a document specification in the input in a server first provider web service descriptor if you know that the resulting WSDL will be used by Axis and .Net clients.

To create a service first provider web service descriptor

1. In the Package Navigator view of Designer, click **File > New > Web Service Descriptor**.
2. In the New Web Service Descriptor dialog box, select the folder in which you want to save the provider web service descriptor. Click **Next**.
3. In the **Element Name** field, specify a name for the provider web service descriptor using any combination of letters, numbers, and/or the underscore character. Click **Next**.
4. Under **Create web service descriptor as**, select **Provider (Inbound Request)**.
5. Under **Web service source**, select **Existing IS service(s)**.

6. Click **Next**.

7. Select one or more services to include as operations in the provider web service descriptor. Click **Next**.

8. Provide the following information:

In this field...	Specify...
SOAP version	Whether SOAP messages for this web service should use **SOAP 1.1** or **SOAP 1.2** message format.
Transport	The transport protocol used to access the web service. Select one of the following: ■ **HTTP** ■ **HTTPS** ■ **JMS**
Use and style for operations	The style/use for operations in the provider web service descriptor. Select one of the following: ■ **Document - Literal** ■ **RPC - Literal** ■ **RPC - Encoded**
Endpoint	The address at which the web service can be invoked. Do one of the following: ■ To use a provider web service endpoint alias to specify the address, select the **Alias** option. Then, in the **Alias** list, select the provider web service endpoint alias. Select DEFAULT(*aliasName*) if you want to use the information in the default provider web service endpoint alias for the address. If the **Alias** list includes a blank row, the Integration Server does not have a default provider web service endpoint alias for the protocol. **Note:** If you select the blank row and a default provider endpoint alias is later set for the selected protocol, Integration Server then uses the information from the alias when constructing the WSDL document and during run-time processing. ■ To specify a host and port as the address, select the **Host** option. Then, in the **Host** field specify the host name for the Integration Server on which the web service resides. In the **Port** field, specify an active

In this field...	Specify...
	HTTP or HTTPS listener port defined on the Integration Server specified in the **Host** field. **Note:** You can only specify **Host** and **Port** for the endpoint if a default provider endpoint alias does not exist for the selected protocol. When a default alias exists, Designer populates the **Host** and **Port** fields with the host and port from the default provider end point alias. **Note:** If you selected JMS as the transport, you must specify an alias. After you select a provider web service endpoint alias, Designer displays the initial portion of the JMS URI that will be used as the address in the **Port address (prefix)** field.
Directive	The SOAP processor used to process the SOAP messages received by the operation in the provider web service descriptor. The Directive list displays all of the SOAP processors registered on the Integration Server. The default processor is **ws - Web Services SOAP Processor**.
Target namespace	The URL that you want to use as the target namespace for the provider web service descriptor. In a WSDL document generated for this provider web service descriptor the elements, attributes, and type definitions will belong to this namespace.

Note:
If you specify a transport, but do not specify a host, port, or endpoint alias, Integration Server uses the primary port as the port in the endpoint URL. If the selected transport and the protocol of the primary port do not match, web service clients will not execute successfully. For more information see "Protocol Mismatch Between Transport and Primary Port" on page 752.

9. Under **Enforce WS-I Basic Profile 1.1 compliance** do one of the following:

   - Select **Yes** if you want Designer to validate all the web service descriptor objects and properties against the WS-I requirements before creating the web service descriptor.
   - Select **No** if you do not want Designer to enforce compliance for WS-I Basic Profile 1.1.

   Note:
   WS-I compliance cannot be enforced if the WSDL contains a SOAP over JMS binding.

10. If you want Integration Server to use the Xerces Java parser to validate the schema elements that represent the signatures of the services used as operations, select the **Validate schema using Xerces** check box.

11. Click **Finish**.

    If Designer cannot create or cannot completely generate a web service descriptor, Designer displays error messages or warning messages.

Notes:

- If you selected the **Validate schema using Xerces** check box, when creating a service first provider web service descriptor, Integration Server converts the signatures of the services used as operations to XML schema elements. Then Integration Server uses the Xerces Java parser to validate the schema elements. If the schema element does not conform syntactically to the schema for XML Schemas defined in *XML Schema Part 1: Structures* (which is located at http://www.w3.org/TR/xmlschema-1), Integration Server does not create the web service descriptor. Instead, Designer displays an error message that lists the number, title, location, and description of the validation errors.

  Note: Integration Server uses Xerces Java parser version J-2.11.0. Limitations for this version are listed at http://xerces.apache.org/xerces2-j/xml-schema.html.

- Set up a package dependency if an IS service uses a document type from a different package as the input or output signature.

- The Message Exchange Pattern (MEP) that Integration Server uses for the operation it creates from the service can be In-Out MEP or In-Only MEP. Integration Server always uses In-Out MEP when the web service descriptor's **Pre-8.2 compatibility mode** property is `true`. When this property is `false`, Integration Server uses:

  - In-Out MEP when the service signature contains both input and output parameters.
  - In-Only MEP when the service signature contains no output parameters.

    Note:
    If you want to use Robust In-Only MEP rather than In-Only MEP, after creating the web service descriptor for a service with no output parameters, add a fault to the operation.

  For more information about Integration Server MEP support, see the *Web Services Developer's Guide*.

Protocol Mismatch Between Transport and Primary Port

A protocol mismatch between the transport for a binder in a provide web service descriptor and the primary port can occur in the following situations:

- When creating a service first web service descriptor, you specify a transport, but do not specify a host, port, or endpoint alias and there is not a default provider endpoint alias for the transport protocol, Integration Server uses the primary port as the port in the endpoint URL.

- When creating a WSDL first provider web service descriptor and default provider endpoint alias is not specified for the protocol used by the binding in the WSDL document. Integration Server uses the primary port as the port in the endpoint URL.

If the selected transport and the protocol of the primary port do not match, Designer displays the following warning when you save the provider web service descriptor:

```
Selected transport protocol does not match that of the primary port
on Integration Server.
```

For example, suppose that you specify a transport of HTTPS when creating the provider web service descriptor, but do not specify a host, port, or endpoint alias. Additionally, Integration Server does not identify a default web service provider endpoint alias for HTTPS. Furthermore, the primary port is an HTTP port. In this situation, Designer displays the above message.

You must resolve this mismatch before making a WSDL document for this provider web service descriptor available to web service consumers. Otherwise, the web service clients will not execute successfully.

Creating a WSDL First Provider Web Service Descriptor

You can create a WSDL first provider web service descriptor from a WSDL document accessed via a URL, from a UDDI registry, or from a service asset in CentraSite.

Keep the following points in mind when creating a WSDL first provider web service descriptor:

- You must have Write access to the folder in which you want to store the provider web service descriptor.
- If the URL for the WSDL contains special characters that need to be encoded, specify the encoding using the **Encoding for WSDL URL** option in the Web Service Descriptor Editor Preferences page.
- Before you can create a provider web service descriptor from a WSDL document that contains a JMS binding, you must have at least one valid web service endpoint alias that specifies the JMS transport. When you create a provider web service descriptor from a WSDL document that specifies a SOAP over JMS binding, Designer automatically assigns the first valid provider web service endpoint alias for JMS to the web service descriptor binder. If there is not valid endpoint alias for JMS, the web service descriptor cannot be created. For example, if the only web service endpoint alias that exists for JMS specifies a SOAP-JMS trigger that no longer exists, Integration Server does not consider the endpoint to be valid and does not create the web service descriptor.
- To create a WSDL first provider web service descriptor from a web service in a UDDI registry, Designer must be configured to connect to that UDDI registry.
- To create a WSDL first provider web service descriptor from a service asset in CentraSite, Designer must be configured to connect to CentraSite.
- You can also create a provider web service descriptor from a service asset in CentraSite by dragging and dropping the service asset from the Registry Explorer view into Package Navigator view. Designer prompts you for a name for the web service descriptor and prompts you to indicate whether you want to create a consumer or provider web service descriptor.
- You can specify whether Integration Server enforces strict, lax, or no content model compliance when generating IS document types from the XML Schema definition contained or referenced

in the WSDL document. Content models provide a formal description of the structure and allowed content for a complex type. The type of compliance that you specify can affect whether Integration Server generates an IS document type from a particular XML Schema definition successfully.

- Do not create a WSDL first provider web service descriptor from a WSDL that specifies RPC-Encoded, contains attributes in its operation signature, and/or has complex type definitions with mixed content. Integration Server might successfully create a web service descriptor from such WSDLs. However, the web service descriptor may exhibit unexpected runtime behavior.

To create a WSDL first provider web service descriptor

1. In Package Navigator view, click **File > New > Web Service Descriptor**.
2. In the New Web Service Descriptor dialog box, select the folder in which you want to save the provider web service descriptor. Click **Next**.
3. In the **Element Name** field, specify a name for the provider web service descriptor using any combination of letters, numbers, and/or the underscore character. Click **Next**.
4. Under **Create web service descriptor as**, select **Provider (Inbound Request)**.
5. Under **Web service source**, select **WSDL**. Click **Next**.
6. Under **Source location**, do one of the following:

Select...	To generate a provider web service descriptor from...
CentraSite	A service asset in CentraSite
File/URL	A WSDL document that resides on the file system or on the Internet.
UDDI	A WSDL document in a UDDI registry

7. Click **Next**.
8. If you selected **CentraSite** as the source, under **Select Web Service from CentraSite**, select the service asset in CentraSite that you want to use to create the web service descriptor. Click **Next**.

   Designer filters the contents of the Services folder to display only service assets that are web services.

   If Designer is not configured to connect to CentraSite, Designer displays the **CentraSite > Connections** preference page and prompts you to configure a connection to CentraSite.
9. If you selected **File/URL** as the source, do one of the following:

- Enter the URL for the WSDL document. The URL should begin with http:// or https://. Click **Next**.
- Click **Browse** to navigate to and select a WSDL document on your local file system. Click **Next**.

10. If you selected **UDDI** as the source, under **Select Web Service from UDDI Registry**, select the web service from the UDDI registry. Click **Next**.

    If Designer is not currently connected to a UDDI registry, the Open UDDI Registry Session dialog box appears. Enter the details to connect to the UDDI registry and click **Finish**.

11. Under **Content model compliance**, select one of the following to indicate how strictly Integration Server enforces content model compliance when creating IS document types from the XML Schema definition in the WSDL document.

Select...	To...
Strict	Generate the IS document type only if Integration Server can represent the content models defined in the XML Schema definition correctly. Document type generation fails if Integration Server cannot accurately represent the content models in the source XML Schema definition. Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type from any XML schema definition that contains those items. **Note:** If Integration Server cannot generate an IS document type that complies with the content model in the XML schema definition in the WSDL document, Integration Server will not generate the provider web service descriptor.
Lax	When possible, generate an IS document type that correctly represents the content models for the complex types defined in the XML schema definition from the WSDL document. If Integration Server cannot correctly represent the content model in the XML Schema definition in the resulting IS document type, Integration Server generates the IS document type using a compliance mode of None. When you select lax compliance, Integration Server will generate the IS document type even if the content models in the XML schema definition cannot be represented correctly.
None	Generate an IS document type that does not necessarily represent or maintain the content models in the source XML Schema definition.

Select...	To...
	When compliance is set to none, Integration Server generates IS document types the same way they were generated in Integration Server releases prior to version 8.2.

12. Select the **Enable MTOM streaming for elements of type base64Binary** check box if you want elements declared to be of type base64Binary in the WSDL or schema to be enabled for streaming of MTOM attachments. For more information about MTOM streaming for web services, see the *Web Services Developer's Guide*.

13. If you want Integration Server to use the Xerces Java parser to validate any schema elements in the WSDL document or any referenced XML Schema definitions before creating the web service descriptor, select the **Validate schema using Xerces** check box.

    Note: Integration Server automatically uses the internal schema parser to validate the schemas in or referenced by a WSDL document. However, the Xerces Java parser provides stricter validation than the Integration Server internal schema parser. As a result, some schemas that the internal schema parser considers to be valid might be considered invalid by the Xerces Java parser. While validation by the Xerces Java parser can increase the time it takes to create a web service descriptor and its associated elements, using stricter validation can help ensure interoperability with other web service vendors.

14. Under **Enforce WS-I Basic Profile 1.1 compliance** do one of the following:

    - Select **Yes** if you want Designer to validate all the WSD objects and properties against the WS-I requirements before creating the WSD.
    - Select **No** if you do not want Designer to enforce compliance for WS-I Basic Profile 1.1.

    Note:
    WS-I Basic Profile 1.0 supports only HTTP or HTTPS bindings. Consequently, WS-I compliance cannot be enforced if the WSDL contains a SOAP over JMS binding.

15. Click **Next** if you want to specify different prefixes than those specified in the XML schema definition. If you want to use the prefixes specified in the XML schema definition itself, click **Finish**.

16. On the Assign Prefixes panel, if you want the web service descriptor to use different prefixes than those specified in the XML schema definition, select the prefix you want to change and enter a new prefix. Repeat this step for each namespace prefix that you want to change.

    Note:
    The prefix you assign must be unique and must be a valid XML NCName as defined by the specification http://www.w3.org/TR/REC-xml-names/#NT-NCName.

17. Click **Finish**.

Designer creates the provider web service descriptor and saves it to the folder you specified. Designer also creates supporting IS elements, such as flow services, IS document types, and IS schemas.

If Designer cannot create or cannot completely generate a provider web service descriptor, Designer displays error messages or warning messages.

18. If Integration Server determines that an XML Schema definition included in or referenced by the WSDL document is invalid or cannot be generated according to the selected content model compliance option, Designer displays the validation error message at the top of the Select Document Type Generation Options panel. Click **Cancel** to abandon this attempt to create a consumer web service descriptor. Alternatively, click **Back** to navigate to previous panels and change your selections.

19. If the WSDL document contains constructs that the current web services stack does not support, Designer displays a message identifying the reasons the web service descriptor cannot be created on the current web services stack. Designer then prompts you to create the web service descriptor using an earlier version of the web services stack. If you want to create the web service descriptor using the earlier version of the web services stack, click **OK**. Otherwise, click **Cancel**.

Notes:

- If the WSDL document contains a construct supported on the web services stack available in Integration Server 7.x, 8.0, and 8.0 SP1 but not on the current web services stack, Designer gives you the option of creating the web service descriptor using the earlier version of the web services stack. If the WSDL document contains any of the following, Designer prompts you to use an earlier version of the web services stack:
  - Mixed “use” values across bindings and operations referenced by services in the WSDL document.
  - Mixed “style” values across bindings referenced by services in the WSDL.
  - More than one operation with the same name in the same port type.
  - Bindings that do not contain all of the operations declared in the port type.
  - Services with multiple bindings that reference different port types.

  Note:
  If you create the web services descriptor using the earlier version of the web services stack, the **Pre-8.2 compatibility mode** property will be set to true for the resulting web service descriptor.
- Integration Server does not create a provider web service descriptor if the WSDL document contains any bindings that are not supported by Integration Server.
- Integration Server will create duplicate operations in case the WSDL document has multiple port names for the same binding. To ensure that duplicate operations are not created, modify the WSDL to make the port name unique for each binding.

- When creating the binders for a WSDL first provider web service descriptor generated from a WSDL document with an HTTP or HTTPS binding, Integration Server assigns the default provider endpoint alias for HTTP or HTTPS to the binder. Integration Server uses the information from the default provider endpoint alias during WSDL generation and run-time processing. Integration Server determines whether to use the HTTP or HTTPS default provider endpoint alias by selecting the default alias for the protocol specified in the soap:addressLocation attribute of the wsdl:port element. If a default provider endpoint alias is not specified for the protocol used by the binding in the WSDL document, Integration Server uses its own hostname as the host and the primary port as the port. If the binding transport protocol is not the same as the primary port protocol, the web service descriptor has a protocol mismatch that you must resolved before making a WSDL generated from the descriptor available to consumers. For more information about a protocol mismatch, see "Protocol Mismatch Between Transport and Primary Port" on page 752.

  Note:
  The default provider endpoint alias also determines security, WS-Addressing, and WS-Reliable Messaging information for the web service descriptor and resulting WSDL document.

- Integration Server uses the internal schema parser to validate the XML schema definition associated with the WSDL document. If you selected the **Validate schema using Xerces** check box, Integration Server also uses the Xerces Java parser to validate the XML Schema definition. With either parser, if the XML Schema does not conform syntactically to the schema for XML Schemas defined in *XML Schema Part 1: Structures* (which is located at http://www.w3.org/TR/xmlschema-1), Integration Server does not create an IS schema or an IS document type for the web service descriptor. Instead, Designer displays an error message that lists the number, title, location, and description of the validation errors within the XML Schema definition.

  Note: Integration Server uses Xerces Java parser version J-2.11.0. Limitations for this version are listed at http://xerces.apache.org/xerces2-j/xml-schema.html.

- When validating XML schema definitions, Integration Server uses the Perl5 regular expression compiler instead of the XML regular expression syntax defined by the World Wide Web Consortium for the XML Schema standard. As a result, in XML schema definitions consumed by Integration Server, the pattern constraining facet must use valid Perl regular expression syntax. If the supplied pattern does not use proper Perl regular expression syntax, Integration Server considers the pattern to be invalid.

  Note:
  If the watt.core.datatype.usejavaregex configuration parameter is set to true, Integration Server uses the Java regular expression compiler instead of the Perl5 regular expression compiler. When the parameter is true, the pattern constraining facet in XML schema definitions must use valid syntax as defined by the Java regular expression.

- When creating the document types for the provider web service descriptor, Integration Server registers each document type with the complex type definition from which it was created in the schema. This enables Integration Server to provide derived type support for document creation and validation.

- If you selected strict compliance and Integration Server cannot represent the content model in the complex type accurately, Integration Server does not generate any IS document types or the web service descriptor.
- The contents of an IS document type with a **Model type** property value other than "Unordered" cannot be modified.
- For an IS document type from a WSDL document, Designer displays the location of the WSDL in the **Source URI** property. Designer also sets the **Linked to source** property to true which prevents any editing of the document type contents. To edit the document type contents, you first need to make the document type editable by breaking the link to the source. However, Software AG does not recommend editing the contents of document types created from WSDL documents.
- If the source WSDL document is annotated with WS-Policy:
  - Integration Server enforces the annotated policy at run time. However, if you attach a policy from the policy repository to the web service descriptor, the attached policy will override the original annotated policy.
  - Integration Server will only enforce supported policy assertions in the annotated policy. For information about supported assertions, see the *Web Services Developer's Guide*.
  - Integration Server does not save the annotated policy in the policy repository.
- The Message Exchange Pattern (MEP) that Integration Server uses for an operation defined in the WSDL can be In-Out MEP, In-Only MEP, or Robust In-Only MEP. Integration Server always uses In-Out MEP when the web service descriptor's **Pre-8.2 compatibility mode** property is set to true. When this property is set to false, Integration Server uses:
  - In-Out MEP when an operation has defined input and output.
  - In-Only MEP when an operation has no defined output and no defined fault.
  - Robust In-Only MEP when an operation has no defined output, but does have a defined fault.

  For more information about Integration Server MEP support, see the *Web Services Developer's Guide*.
- If the WSDL is annotated with WS-Policy, Integration Server will only enforce supported policy assertions. Currently Integration Server supports only WS-Security policies. Also be aware that Integration Server does not save the WS-Policy that is in the WSDL in the policy repository. Integration Server will enforce the annotated policy unless a policy that resides in the Integration Server policy repository is specifically attached to the web service descriptor. If you attach a policy to the web service descriptor, the attached policy will override the original annotated policy.
- Integration Server creates the docTypes and services folders to store the IS document types, IS schemas, and skeleton services generated from the WSDL document. These folders are reserved for elements created by Integration Server for the web service descriptor only. Do not place an custom IS elements in these folders. During refresh of a web service descriptor, the contents of these folders will be deleted and recreated.

- If an XML Schema definition referenced in the WSDL document contains the <!DOCTYPE declaration, Integration Server issues a java.io.FileNotFoundException. To work around this issue, remove the <!DOCTYPE declaration from the XML Schema definition.
- When creating a WSDL first provider web service descriptor from an XML Schema definition that imports multiple schemas from the same target namespace, Integration Server throws Xerces validation errors indicating that the element declaration, attribute declaration, or type definition cannot be found. The Xerces Java parser honors the first <import> and ignores the others. To work around this issue, you can do one of the following:
  - Combine the schemas from the same target namespace into a single XML Schema definition. Then change the XML schema definition to import the merged schema only.
  - When creating the WSDL first provider web service descriptor, clear the **Validate schema using Xerces** check box to disable schema validation by the Xerces Java parser. When generating the web service descriptor, Integration Server will not use the Xerces Java parser to validate the schemas associated with the XML Schema definition.

About Consumer Web Service Descriptors

To use Integration Server and Designer to invoke web services located on remote servers, create a consumer web service descriptor from a WSDL document, from a service asset in CentraSite, or from the web service in the UDDI registry. A *consumer web service descriptor (WSD)* defines an external web service. It contains all the data from the WSDL document that defines the web service, as well as data needed for certain Integration Server run-time properties.

Integration Server creates a *web service connector (WSC)* for each operation in the web service. A web service connector is a flow service with an input and output signature that corresponds to the input and output messages of the web service operation.

The web service connector is a flow service that:

- Uses an input and output signature that corresponds to the input and output messages of the web service operation.
- Contains flow steps that create and send a message to the web service using the transport, protocol, and location information specified in the web service.
- Contains flow steps that extract data from the output message returned by the web service.

When Integration Server executes a web service connector, the web service connector calls a specific operation of a web service.

In versions 9.0 and later, Integration Server also creates a response service for each operation in the WSDL document. Response services are flow services to which you can add custom logic to process asynchronous SOAP responses. For more information about response services, see "About Response Services" on page 770.

Creating a Consumer Web Service Descriptor

You can create a consumer WSD from a WSDL document accessible via a URL, a WSDL document in a UDDI registry, or a service asset in CentraSite.

- You must have Write access to the folder in which you want to store the consumer web service descriptor.
- If you are creating a consumer web service descriptor from a WSDL located on a website, if the website on which the document resides is password protected, you must download the WSDL document to your local file system and then create the consumer web service descriptor.
- If the URL for the WSDL contains special characters that need to be encoded, specify the encoding using the **Encoding for WSDL URL** option in the Web Service Descriptor Editor Preferences page.
- To create a consumer web service descriptor from a service asset in CentraSite, Designer must be configured to connect to CentraSite.
- You can also create a consumer web service descriptor from a service asset in CentraSite by dragging and dropping the service asset from the Registry Explorer view into Package Navigator view. Designer prompts you for a name for the web service descriptor and prompts you to indicate whether you want to create a consumer or provider web service descriptor.
- To create a consumer web service descriptor from a web service in a UDDI registry, Designer must be configured to connect to that UDDI registry.
- You can specify whether Integration Server enforces strict, lax, or no content model compliance when generating IS document types from the XML Schema definition contained or referenced in the WSDL document. Content models provide a formal description of the structure and allowed content for a complex type. The type of compliance that you specify can affect whether Integration Server generates an IS document type from a particular XML Schema definition successfully.
- Do not create a consumer web service descriptor from a WSDL that specifies RPC-Encoded, contains attributes in its operation signature, and/or has complex type definitions with mixed content. Integration Server might successfully create a web service descriptor from such WSDLs. However, the web service descriptor may exhibit unexpected runtime behavior.

To create a consumer web service descriptor

1. In Package Navigator view, click **File > New > Web Service Descriptor.**
2. In the New Web Service Descriptor dialog box, select the folder in which you want to save the consumer web service descriptor. Click **Next**.
3. In the **Element Name** field, specify a name for the consumer WSD using any combination of letters, numbers, and/or the underscore character. Click **Next**.

4. Under **Create web service descriptor as**, select **Consumer (Outbound Request)**.

5. Under **Web service source**, select **WSDL**. Click **Next**.

6. Under **Source location**, do one of the following:

Select...	To generate a consumer web service descriptor from...
CentraSite	A service asset in CentraSite
File/URL	A WSDL document that resides on the file system or on the Internet.
UDDI	A WSDL document in a UDDI registry

7. If you specified **CentraSite** as the source, under **Select web service from CentraSite**, select the service asset in CentraSite that you want to use to create the web service descriptor. Click **Next**.

   Designer filters the contents of the Services folder to display only service assets that are web services.

   If Designer is not configured to connect to CentraSite, Designer displays the **CentraSite > Connections** preference page and prompts you to configure a connection to CentraSite.

8. If you specified **File/URL** as the source, do one of the following:

   - Enter the URL for the WSDL document. The URL should begin with http:// or https://. Click **Next**
   - Click **Browse** to navigate to and select a WSDL document on your local file system. Click **Next**

9. If you specified **UDDI** as the source, under **Select web service from UDDI Registry**, select the web service from the UDDI registry. Click **Next**.

   If Designer is not currently connected to a UDDI registry, the Open UDDI Registry Session dialog box appears. Enter the details to connect to the UDDI registry and click **Finish**.

10. Under **Content model compliance**, select one of the following to indicate how strictly Integration Server enforces content model compliance when creating IS document types from the XML Schema definition in the WSDL document.

Select...	To...
Strict	Generate the IS document type only if Integration Server can represent the content models defined in the XML Schema definition correctly. Document type generation fails if Integration Server cannot accurately represent the content models in the source XML Schema definition.

Select...	To...
	Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type from any XML schema definition that contains those items. **Note:** If Integration Server cannot generate an IS document type that complies with the content model in the XML schema definition in the WSDL document, Integration Server will not generate the consumer web service descriptor.
Lax	When possible, generate an IS document type that correctly represents the content models for the complex types defined in the XML schema definition from the WSDL document. If Integration Server cannot correctly represent the content model in the XML Schema definition in the resulting IS document type, Integration Server generates the IS document type using a compliance mode of None. When you select lax compliance, Integration Server will generate the IS document type even if the content models in the XML schema definition cannot be represented correctly.
None	Generate an IS document type that does not necessarily represent or maintain the content models in the source XML Schema definition. When compliance is set to none, Integration Server generates IS document types the same way they were generated in Integration Server releases prior to version 8.2.

11. Under **Document type generation**, select the **Enable MTOM streaming for elements of type base64Binary** check box if you want elements declared to be of type base64Binary in the WSDL or schema to be enabled for streaming of MTOM attachments. For more information about MTOM streaming for web services, see the *Web Services Developer's Guide*.

12. If you want to use the Xerces Java parser to validate any schema elements in the WSDL document or any referenced XML Schema definitions before creating the web service descriptor, select the **Validate schema using Xerces** check box.

    Note: Integration Server uses an internal schema parser to validate the schemas in or referenced by a WSDL document. However, the Xerces Java parser provides stricter validation than the Integration Server internal schema parser. As a result, some schemas that the internal schema parser considers to be valid might be considered invalid by the Xerces Java parser. While validation by the Xerces Java parser can increase the time it takes to create a web service descriptor and its associated elements, using stricter validation can help ensure interoperability with other web service vendors.

13. Under **Enforce WS-I Basic Profile 1.1 compliance** do one of the following:

- Select **Yes** if you want Designer to validate all the WSD objects and properties against the WS-I requirements before creating the WSD.
- Select **No** if you do not want Designer to enforce compliance for WS-I Basic Profile 1.1.

Note:
WS-I Basic Profile 1.0 supports only HTTP or HTTPS bindings. Consequently, WS-I compliance cannot be enforced if the WSDL contains a SOAP over JMS binding.

14. Click **Next** if you want to specify different prefixes than those specified in the XML schema definition. If you want to use the prefixes specified in the XML schema definition itself, click **Finish**.

15. On the Assign Prefixes panel, if you want the web service descriptor to use different prefixes than those specified in the XML schema definition, select the prefix you want to change and enter a new prefix. Repeat this step for each namespace prefix that you want to change.

    Note:
    The prefix you assign must be unique and must be a valid XML NCName as defined by the specification http://www.w3.org/TR/REC-xml-names/#NT-NCName.

16. Click **Finish**.

    Designer creates the consumer web service descriptor and saves it to the specified folder. Designer also creates supporting IS elements, such as web service connectors, IS document types, and response services and places them in the same folder. For more information about what elements Integration Server creates, see "Supporting Elements for a Consumer Web Service Descriptor" on page 767.

    If Designer cannot create or cannot completely generate a consumer WSD, Designer displays error messages or warning messages.

    If Integration Server determines that an XML Schema definition included in or referenced by the WSDL document is invalid or cannot be generated according to the selected content model compliance option, Designer displays the validation error message at the top of the Select Document Type Generation Options panel. Click **Cancel** to abandon this attempt to create a consumer web service descriptor. Alternatively, click **Back** to navigate to previous panels and change your selections.

Notes:

- If the WSDL document contains a construct supported on the web services stack available in Integration Server 7.x, 8.0, and 8.0 SP1 but not on the current web services stack, Designer gives you the option of creating the web service descriptor using the earlier version of the web services stack. If the WSDL document contains any of the following, Designer prompts you to use an earlier version of the web services stack:
  - Mixed "use" values across bindings and operations referenced by services in the WSDL document.
  - Mixed "style" values across bindings referenced by services in the WSDL.

- More than one operation with the same name in the same port type.
- Bindings that do not contain all of the operations declared in the port type.
- Services with multiple bindings that reference different port types.

Note:
If you create the web services descriptor using the earlier version of the web services stack, the **Pre-8.2 compatibility mode** property will be set to true for the resulting web service descriptor.

- Integration Server does not create binders for unsupported bindings in the WSDL document. If the WSDL document does not contain any bindings supported by Integration Server, Integration Server does not create a consumer web service descriptor.
- When creating the document types for the consumer web service descriptor, Integration Server registers each document type with the complex type definition from which it was created in the schema. This enables Integration Server to provide derived type support for document creation and validation.
- Integration Server uses the internal schema parser to validate the XML schema definition associated with the WSDL document. If you selected the **Validate schema using Xerces** check box, Integration Server also uses the Xerces Java parser to validate the XML Schema definition. With either parser, if the XML Schema does not conform syntactically to the schema for XML Schemas defined in *XML Schema Part 1: Structures* (which is located at http://www.w3.org/TR/xmlschema-1), Integration Server does not create an IS schema or an IS document type for the web service descriptor. Instead, Designer displays an error message that lists the number, title, location, and description of the validation errors within the XML Schema definition.

  Note: Integration Server uses Xerces Java parser version J-2.11.0. Limitations for this version are listed at http://xerces.apache.org/xerces2-j/xml-schema.html.
- When validating XML schema definitions, Integration Server uses the Perl5 regular expression compiler instead of the XML regular expression syntax defined by the World Wide Web Consortium for the XML Schema standard. As a result, in XML schema definitions consumed by Integration Server, the pattern constraining facet must use valid Perl regular expression syntax. If the supplied pattern does not use proper Perl regular expression syntax, Integration Server considers the pattern to be invalid.

  Note:
  If the watt.core.datatype.usejavaregex configuration parameter is set to true, Integration Server uses the Java regular expression compiler instead of the Perl5 regular expression compiler. When the parameter is true, the pattern constraining facet in XML schema definitions must use valid syntax as defined by the Java regular expression.
- If you selected strict compliance and Integration Server cannot represent the content model in the complex type accurately, Integration Server does not generate any IS document types or the web service descriptor.

- For an IS document type from a WSDL document, Designer displays the location of the WSDL in the **Source URI** property. Designer also sets the **Linked to source** property to true which prevents any editing of the document type contents. To edit the document type contents, you first need to make the document type editable by breaking the link to the source. However, Software AG does not recommend editing the contents of document types created from WSDL documents.
- The contents of an IS document type with a **Model type** property value other than "Unordered" cannot be modified.
- Operations and binders cannot be added, edited, or removed from a consumer web service descriptor.
- The Message Exchange Pattern (MEP) that Integration Server uses for an operation defined in the WSDL can be In-Out MEP, In-Only MEP, or Robust In-Only MEP. Integration Server always uses In-Out MEP when the web service descriptor's **Pre-8.2 compatibility mode** property is `true`. When this property is `false`, Integration Server uses:
  - In-Out MEP when an operation has defined input and output.
  - In-Only MEP when an operation has no defined output and no defined fault. The web service connector that Integration Server creates will no SOAP message-related output parameters and, when executed, will not return output related to a SOAP response.
  - Robust In-Only MEP when an operation has no defined output, but has a defined fault. The web service connector that Integration Server creates will return no output related to a SOAP response if the operation executes successfully. However, if an exception occurs, the web service connector returns the SOAP fault information as output.

  For more information about Integration Server MEP support, see the *Web Services Developer's Guide*.
- Integration Server creates response services for all In-Out and Robust In-Only MEP operations in the WSDL document.
- When creating a web service descriptor from a WSDL document, Integration Server treats message parts that are defined by the type attribute instead of the element attribute as an error and does not allow the web service descriptor to be created. You can change this behavior by setting the watt.server.SOAP.warnOnPartValidation parameter to `true`. When this parameter is set to `true`, Integration Server will return a warning instead of an error and will allow the web service descriptor to be created.
- If the WSDL document is annotated with WS-Policy:
  - Integration Server enforces the annotated policy at run time. However, if you attach a policy from the policy repository to the web service descriptor, the attached policy will override the original annotated policy.
  - Integration Server will only enforce supported policy assertions in the annotated policy. Currently Integration Server supports only WS-Security policies.
  - Integration Server does not save the annotated policy in the policy repository.

- If an XML Schema definition referenced in the WSDL document contains the <!DOCTYPE declaration, Integration Server issues a java.io.FileNotFoundException. To work around this issue, remove the <!DOCTYPE declaration from the XML Schema definition.
- When creating a consumer web service descriptor from an XML Schema definition that imports multiple schemas from the same target namespace, Integration Serverthrows Xerces validation errors indicating that the element declaration, attribute declaration, or type definition cannot be found. The Xerces Java parser honors the first <import> and ignores the others. To work around this issue, you can do one of the following
  - Combine the schemas from the same target namespace into a single XML Schema definition. Then change the XML schema definition to import the merged schema only.
  - When creating the consumer web service descriptor, clear the **Validate schema using Xerces** check box to disable schema validation by the Xerces Java parser. When generating the web service descriptor, Integration Server will not use the Xerces Java parser to validate the schemas associated with the XML Schema definition.

Supporting Elements for a Consumer Web Service Descriptor

When Designer creates a consumer web service descriptor, it also creates supporting IS elements. Each of the IS elements that Designer creates corresponds to an element in the WSDL document.

This IS element...		Corresponds to this WSDL element...
	*consumerWsdName_*	The WSDL document. All supporting IS elements for the consumer web service descriptor are contained in this new folder. The folder name is the same as the web service descriptor, with a suffix of an "_" (underscore).
	docType folder	All of the IS document types generated from the messages in the WSDL document.
	connectors folder	All of the web service connectors generated from the operations in the WSDL document.
	responseServices folder (Available in Integration Server version 9.0 and later only.)	All of the response services generated from the operations in the WSDL document and the genericFault_Response service.
	Web service connector	Each unique `<operation>` element in a `<portType>` element; the web service connector name corresponds to the portType name and operation name.
	IS document type	Each `<message>` element in the WSDL document. The IS document type name corresponds to the message name.

This IS element...		Corresponds to this WSDL element...
	IS schema	Each target namespace to which the element declarations, attribute declarations, and type definitions that define the message parts (input and output signature) belong. **Note:**Integration Server assigns any IS schemas to a unique schema domain for that web service descriptor.
	Response service (Available in Integration Server version 9.0 and later only.)	Each unique `<operation>` element in a `<portType>` element; the response service name corresponds to the portType name and operation name, with a suffix of "_Response.
	genericFault_ Response service	The default response service that Integration Server invokes when Integration Server cannot determine the specific response service for an asynchronous SOAP response or if there are errors while processing the response.

Note:
The *consumerWSDName_* folder and its subfolders docTypes, connectors, and responseServices are reserved for elements created by Integration Server for the web service descriptor only. Do not place any custom IS elements in these folders.

About Web Service Connectors

A web service connector is a flow service that Integration Server creates at the time it creates the consumer web service descriptor. A web service connector contains the information and logic needed to invoke an operation defined in the WSDL document used to create the consumer web service descriptor.

When creating a consumer web service descriptor from a WSDL document, Integration Server creates a web service connector for each operation and portType combination contained in the WSDL document. For example, if a WSDL document contains two portType declarations and each portType contains three operations, Integration Server creates six web service connectors.

A web service connector:

- Uses an input and output signature that corresponds to the input message, output message, and headers defined for the operation in the WSDL document. The web service connector signature also contains optional inputs that you can use to control the execution of logic in the web service connector.
- Represents a SOAP fault structure in the output signature differently based on the version of the Integration Server on which the web service descriptor is created. To learn more about the output signature of a web service connector, see *Web Services Developer's Guide*.

- Contains flow steps that create and send a message to the web service endpoint using the transport, protocol, and location information specified in the web service's WSDL document in conjunction with input supplied to the web service connector.
- Contains flow steps that extract data or fault information from the response message returned by the web service.

Important:
Do not edit the flow steps in a web service connector.

Note:
A web service connector that worked correctly with previous versions of Developer, Designer, and Integration Server should continue to work with version 8.2 and later. In addition, any external clients created from WSDL generated from previous versions of Developer and Integration Server should continue to work as they did in the previous version.

For detailed information about a web service connector, such as a description of the web service connector signature, see the *Web Services Developer's Guide*.

Refreshing a Web Service Connector

When you create a consumer web service descriptor, Designer automatically generates the web service connector(s). You must refresh the web service connectors after you have added, deleted, or modified any of the following.

- Header
- Fault
- Endpoint alias within a binder for a consumer web service descriptor

Refreshing the web service connectors overwrites all the existing web service connectors for a consumer web service descriptor.

Keep the following points in mind when refreshing a web service connector:

- When refreshing a web service connector, Integration Server deletes the *consumerWSDName_* folder and all elements contained in that folder and its subfolders. Integration Server will not recreate any elements manually added to the folder or its subfolders. Integration Server will not recreate modifications made to any of the original elements in the *consumerWSDName_* folder.
- Refreshing a web service connector does not change the structure of the fault in the output signature. That is, when you refresh a web service connector for a web service descriptor created using an Integration Server version prior to 8.2, the web service connector output signature retains the SOAP fault document that is specific to the SOAP protocol (i.e., SOAP 1.1 or SOAP 1.2). Similarly, if you refresh a web service connector for a web service descriptor created using Integration Server 8.2, the web service connector's output signature will continue to have the generic SOAP fault structure. For more information about how the output signature of web service connector depends on the version of the Integration Server on which the web service descriptor is created, see *Web Services Developer's Guide*.

- If the **Validate Schema using Xerces** property is set to true for a web service descriptor, Integration Server validates the schemas associated with a consumer web service descriptor when you refresh the web service connector.
- Refreshing web service connectors is different than refreshing a web service descriptor. When refreshing web service connectors, Integration Server uses the original WSDL document to recreate the web service connectors and the contents of the *consumerWSDName_* folder. When refreshing a web service descriptor, Integration Server uses an updated version of the WSDL document to regenerate the web service descriptor and its associated IS elements. For more information about refreshing web service descriptors, see "About Refreshing a Web Service Descriptor" on page 771.
- If you are using the local service development feature, using versions of Subversion prior to 1.7 as your VCS client might cause issues while refreshing web service connectors. Software AG recommends that you use Subversion 1.7 or higher as your VCS client.

To refresh a web service connector

1. In Package Navigator view, open the consumer WSD for which you want to refresh web service connectors.
2. Click the Operations tab or the Binders tab.
3. Click or right-click and select **Refresh Web Service Connectors.**

   Integration Server regenerates *all* web service connectors in the consumer WSD, overwriting the existing web service connectors.

Invoking a Web Service Using a Web Service Connector

To invoke a web service, or more specifically, an operation in a web service, create a flow service that invokes the web service connector that corresponds to the operation you want to use. Because a web service connector is a flow service, you invoke the web service connector in the same way in which you would a regular flow service.

Note:
If the web service connector uses a JMS binding to send a message using SOAP over JMS, you can specify how Integration Server proceeds when the JMS provider is not available at the time the message is sent. For more information, see "Configuring Use of the Client Side Queue" on page 798.

About Response Services

In versions 9.0 and later, Integration Server creates a responseServices folder along with connectors and docTypes folders when you create a consumer web service descriptor.

The responseServices folder contains a response service for each In-Out and Robust-In-Only MEP operation in the WSDL document from which the consumer web service descriptor is created.

Response services are flow services to which you can add custom logic to process asynchronous SOAP responses. Integration Server creates the response services only if the consumer web service descriptor:

- Is created on Integration Server version 9.0 or later.
- Has the **Pre-8.2 compatibility mode** property set to false.

Integration Server invokes the response services for processing asynchronous SOAP responses received for the associated consumer web service descriptor. That is, Integration Server invokes a response service when Integration Server receives a SOAP response with the endpoint URL pointing to a consumer web service descriptor and if this SOAP response contains a WS-Addressing action through which the response service can be resolved.

The responseServices folder also contains a genericFault_Response service, which is the default response service that Integration Server invokes when Integration Server cannot determine the specific response service for a SOAP response or if there are errors while processing the response.

For more information about response services and how Integration Server processes responses asynchronously, see the *Web Services Developer's Guide*

About Refreshing a Web Service Descriptor

If the WSDL document used to create a web service descriptor changes, you may want to refresh the web service descriptor to reflect the recent changes. For example, if you created a WSDL first provider web service descriptor from a WSDL document that has since changed to include a new operation or new input/output messages, you can refresh the web service descriptor. Refreshing the web service descriptor does the following:

- Updates the web service descriptor or its associated IS elements to reflect changes in existing elements in the updated WSDL document.
- Adds elements, such as operations, headers, or binders, to the web service descriptor to reflect new elements in the updated WSDL document.
- Adds new IS elements, such as IS document types, IS schemas, and services, that correspond to new elements in the updated WSDL document.
- Removes web service descriptor elements or IS elements that correspond to elements that have been removed from the updated WSDL document.
- Preserves any changes you made to the web service descriptor since it was created from the original WSDL document.

Refreshing a web service descriptor is different than refreshing web service connectors. When refreshing a web service descriptor, Integration Server uses an updated version of the WSDL document to regenerate the web service descriptor and its associated IS elements. When refreshing web service connectors, Integration Server uses the original WSDL document to recreate the web service connectors and the contents of the *consumerWSDName_* folder. For more information about refreshing web service connectors, see "Refreshing a Web Service Connector" on page 769.

The following table provides an overview of the activities involved in refreshing a web service descriptor.

Step	Description
1	You select the web service descriptor that you want to refresh. You can refresh WSDL first provider web service descriptors or consumer web service descriptors created on Integration Server version 7.1 or later. **Note:** Service first provider web service descriptors are not created from a WSDL document and therefore cannot be refreshed.
2	You specify the location of the WSDL document that you want Integration Server to use when refreshing the web service descriptor. If the web service descriptor was created on Integration Server version 8.2 or later, Integration Server defaults to the WSDL document at the location specified by the **Source URI** property. If the web service descriptor was created before Integration Server version 8.2, Designer prompts you to select the location of the WSDL document to use as the source for the refresh.
3	Integration Server creates a backup copy of the web service descriptor and its associated elements, such as IS document types, IS schemas, services, and web service connectors. Integration Server uses the most recently saved version of the web service descriptor and its associated elements as the backup copy. Integration Server makes a backup copy in case it cannot refresh the web service descriptor successfully.
4	Integration Server regenerates the web service descriptor using the options specified in the New Web Service Descriptor wizard at the time the web service descriptor was first created. For example, if you specified **Strict** for the **Content model compliance** option, Integration Server uses the **Strict** option when refreshing. **Note:** If you want Integration Server to use different options than the original ones when refreshing the web service descriptor, do not refresh the web service descriptor. Instead, delete the web service descriptor and then recreate it using the updated WSDL document and the different New Web Service Descriptor wizard options.
5	During regeneration, Integration Server does the following: ■ If an existing element in the WSDL document has been modified, Integration Server updates the corresponding elements in the web service descriptor or its associated IS elements. For example, suppose that a complex type definition changed and that type definition was used in the input message for an operation. During

Step	Description
	refresh, Integration Server would recreate the document type that corresponds to the input signature. ■ If the WSDL document includes new information or elements, Integration Server adds that information to the web service descriptor or one if its associated IS elements. For example, if the WSDL used to create a provider web service descriptor includes a new operation, Integration Server generates a new skeleton service and adds the service as an operation in the web service descriptor. ■ If information or an element has been removed from the updated WSDL document, Integration Server removes the corresponding web service descriptor information or associated IS element. For example, suppose that the original WSDL document included a header but the updated WSDL document does not include that header. During refresh, Integration Server removes the header from the web service descriptor and deletes the IS document type that corresponds to the original header. **Note:**Integration Server considers a renamed element to be a new element. For example if the name of an operation changed from "myOperation" to "yourOperation", Integration Server removes the operation "myOperation" from the web service descriptor. Integration Server creates a new service for "yourOperation" and adds that service to the web service descriptor as an operation. ■ Integration Server merges in changes that you made since the web service descriptor was first created or since the last refresh. For example, if you added logic to a skeleton service generated for a WSDL first provider web service descriptor, Integration Server adds that logic to the refreshed service. If you added a header or fault to the web service descriptor, Integration Server adds that header or fault to the refreshed web service descriptor. **Note:** When refreshing a web service descriptor, Integration Server does not preserve any changes made to IS document types or IS schemas that were generated from the WSDL document. ■ Integration Server sets the properties of the refreshed web service descriptor to match the property values that were specified before the web service descriptor was refreshed. Integration Server also ensures that any services created from the original WSDL document have the same properties after the refresh.
6	Upon successful refresh of the web service descriptor, Integration Server deletes the backup copy of the web service descriptor and its associated

Step	Description
	elements. If Integration Server cannot successfully refresh the web service descriptor, Integration Server reverts to the backup copy.

How Refresh Affects a Web Service Descriptor

Integration Server handles the updating of each web service descriptor element or associated IS element differently depending on:

- The web service descriptor element, such as an operation, binder, or header.
- The type IS element.
- Whether the web service descriptor element or IS element changed since the web service descriptor was first created.
- Whether the updated WSDL document contains an element that corresponds to the web service descriptor element or IS element.

The following table provides details about how Integration Server handles specific IS elements during refresh.

For this element...	During refresh Integration Server...
IS document type	Deletes all of the document types thatIntegration Server generated from the WSDL document. Integration Server then creates new document types using the updated WSDL document. Any changes made to the original document types will be lost.
IS schema	Deletes all of the IS schemas that Integration Server generated from the WSDL document. Integration Server then creates new IS schemas using the updated WSDL document. Any changes made to the original IS schemas will be lost.
Service	Does one of the following for the skeleton services generated for operations in the original WSDL document: ■ If logic has been added to the skeleton service or service properties have been set and the corresponding operation exists in the updated WSDL document, Integration Server merges the logic into the refreshed service and ensures that the property values match the values set prior to refreshing. ■ If logic has not been added to the skeleton service, service properties have not been set, and the corresponding operation exists in the updated WSDL document, Integration Server recreates the empty skeleton service.

For this element...	During refresh Integration Server...
	■ If a service corresponds to an operation that does not exist in the updated WSDL document, Integration Server removes the operation that corresponds to the service from the web service descriptor. Integration Server keeps the service in the "services" folder. **Note:**Integration Server considers a renamed operation to be a new operation.
Web service connector	Deletes and recreates all web service connectors. Any changes made to a web service connector, including changes for pipeline mapping, will be lost.
connectors folder	Deletes the connectors folder and all elements contained in that folder and its subfolders. Integration Server will not recreate any elements manually added to the folder or its subfolders.
*consumerWSDName_* folder	Deletes the *consumerWSDName_* folder and all elements contained in that folder and its subfolders. Integration Server will not recreate any elements manually added to the folder or its subfolders. Integration Server will not recreate modifications made to any of the original elements in the *consumerWSDName_* folder.
docTypes folder	Deletes the docTypes folder and all elements contained in that folder and its subfolders. Integration Server will not recreate any elements manually added to the folder or its subfolders.
responseServices folder	Does the following with the contents of the responseServices folder: ■ For new operations in the updated WSDL document, Integration Server adds new response services. ■ For modified operations, Integration Server updates the response services including merging in any logic that was added. ■ For deleted operations, Integration Server removes the operation that corresponds to the service from the web service descriptor. Integration Server keeps the response service in the "responseServices" folder.
services folder	Does the following with the contents of the services folder: ■ Adds new skeleton services for new operations in the updated WSDL document. ■ For modified operations, updates the skeleton services including merging in any logic that was added. For more information, see the "Service" row in this table.

For this element...	During refresh Integration Server...
	■ For deleted operations, Integration Server removes the operation that corresponds to the service from the web service descriptor. Integration Server keeps the service in the “services” folder.

The following table provides details about how refreshing a web service descriptor affects the contents of the web service descriptor itself.

For this web service descriptor element	During refresh Integration Server...
Binders	Updates the binders to reflect any new port information in the WSDL document. Integration Server also updates the operations for each binder to reflect any new or removed operations in the updated WSDL document. Integration Server updates the SOAP action values assigned to operations in binders to reflect any changes in the updated WSDL document.
Header/Fault	For a header/fault defined in the WSDL document, Integration Server does one of the following: ■ If the updated WSDL document does not contain the header/fault, Integration Server removes the header/fault from the web service descriptor. Integration Server also deletes the document types used to define the header/fault documents from the docTypes folder. ■ If the updated WSDL document contains the header/fault, Integration Server keeps the header/fault in the web service descriptor. Integration Server recreates the document types used to define the header/fault documents and places them in the docTypes folder. ■ If the updated WSDL contains a new header/fault, Integration Server adds the header/fault to the web service descriptor. Integration Server adds the new document types used to define the header/fault documents to the docTypes folder. For a header/fault that was added manually after the web service descriptor was created, Integration Server adds the document types that define the header/fault to the refreshed web service descriptor. **Note:** If a header/fault that was added manually after the web service descriptor was created has the same name as a header/fault in the updated WSDL document, Integration Server replaces the manually added header/fault with the

For this web service descriptor element	During refresh Integration Server...
	header/fault that Integration Server generates programmatically from the WSDL document. This might result in broken mappings or unexpected behavior in the handler service associated with the header/fault.
Handler	Integration Server adds handlers defined in the previous version of the web service descriptor to the refreshed web service descriptor.
Operations	Updates the operations to reflect any new or removed operations in the updated WSDL document.
Policy	Attaches any polices added in the previous version of the web service descriptor to the refreshed version of the web service descriptor.

Considerations for Refreshing a Web Service Descriptor

Because refreshing a web service descriptor involves deleting, recreating, and merging elements, it is possible that refreshing will result in broken mappings, services that do not execute to completion successfully, and other issues that need to be resolved. Before refreshing a web service descriptor, review the following considerations.

- Refresh a web service descriptor only if you are familiar with the original WSDL document, the changes in the updated WSDL document, and the web service descriptor. Designer does not provide a list of changes to the web service descriptor as part of the refresh. You will need to use your knowledge of the WSDL document changes and the web service descriptor to ensure that operations, services, pipeline mapping, and other aspects of the web service descriptor work as expected.
- During refresh, mappings between variables might break or be lost. This is particularly true when the web service descriptor has manually added headers or faults and the updated WSDL document has new headers or faults of the same name.
- During refresh of a consumer web service descriptor, Integration Server deletes and recreates the contents of the *consumerWSDName_*. This includes all of the document types, Integration Server schemas and web service connectors generated from the original WSDL document. Any changes made to these elements will be lost. For web service connectors, this includes maps (links) between variables in the pipeline, variables added to the pipeline, variables dropped from the pipeline, and values assigned to pipeline variables.
- During refresh of a WSDL first provider web service descriptor, Integration Server deletes and recreates the contents of the docTypes folder. Changes made to the IS document types and IS schemas generated from the original WSDL document will be lost.
- Because Integration Server deletes and recreates the contents of the *consumerWSDName_* folder, docTypes folder, connectors folder, and services folder during refresh, do not place any custom

elements in these folders. These folders are reserved for elements created by Integration Server for the web service descriptor only. Before refreshing a web service descriptor, remove any custom elements from these folders.

- If you used an IS element created by Integration Server for the web service descriptor with another IS element that is not associated with the web service descriptor, refreshing the web service descriptor might break the other usages of the IS element. For example, suppose that you used an IS document type created for an input message as the input signature of a service not used as an operation in the web service descriptor. If the input messages is removed from the updated WSDL document upon refresh, the other service will have a broken reference. The service will reference a document type that no longer exists.
- If you refresh a WSDL first provider web service descriptor for which web service clients have already been created, the web service clients will need to be recreated. Consumers will need to recreate their web service client using the new WSDL document that Integration Server generates for the provider web service descriptor.
- During refresh, Integration Server regenerates the web service descriptor using the functionality and features available in the Integration Server version on which the web service descriptor was originally created. After refreshing the web service descriptor, the **Created on version** property value is the same version of Integration Server as before the refresh. Refreshing a web service descriptor on the latest version of Integration Server does not update the web service descriptor to include all the web service features and functionality available in the current version of Integration Server. If you want the web service descriptor to use the features available with the current version of Integration Server, delete the web service descriptor and recreate it using Designer and the current version of Integration Server.
- If you are using the local service development feature, using versions of Subversion prior to 1.7 as your VCS client might cause issues while refreshing web service connectors. Software AG recommends that you use Subversion 1.7 or higher as your VCS client.

Refreshing a Web Service Descriptor

To update a web service descriptor to use the latest version of a WSDL document, you can refresh the web service descriptor. Keep the following points in mind when you refresh a web service descriptor.

- You can refresh any consumer web service descriptor or WSDL first provider web service descriptor created on version 7.1 or later.
- To refresh a web service descriptor, you do not need to have Write access to the web service descriptor or any of its associated elements (document types, schemas, services, or web service connectors).
- When refreshing a web service descriptor Integration Server uses the same options you selected in the New Web Service Descriptor wizard when you originally created the web service descriptor. If you want to use different options, you must delete the web service descriptor and recreate it using the updated WSDL document.

- If the **Validate Schema using Xerces** property is set to true for a web service descriptor, Integration Server validates the schemas associated with the web service descriptor during refresh.
- Any pre-requisites that existed for generating the original web service descriptor apply to refreshing the web service descriptor. This includes, but is not limited to the following:
  - To refresh a web service descriptor whose source is in a UDDI registry, Designer must be configured to connect to that UDDI registry.
  - To refresh a web service descriptor whose source is a service asset in CentraSite, Designer must be configured to connect to CentraSite.
  - Before you can refresh a web service descriptor from a WSDL document that contains a JMS binding, you must have at least one valid web service endpoint alias that specifies the JMS transport.

  For information about pre-requisites for creating a WSDL first provider web service descriptor, see "Creating a WSDL First Provider Web Service Descriptor" on page 753. For information about pre-requisites for creating a consumer web service descriptor, see "Creating a Consumer Web Service Descriptor" on page 761.
- Refreshing a web service descriptor is different than refreshing web service connectors. For more information about refreshing web service connectors, see "Refreshing a Web Service Connector" on page 769.
- Before refreshing a web service descriptor, review the information in "Considerations for Refreshing a Web Service Descriptor" on page 777.

To refresh a web service descriptor

1. In Package Navigator view, lock the web service descriptor that you want to refresh.
2. Right-click the web service descriptor and select **Refresh Web Service Descriptor**.
3. Review the informational message about potential changes to the existing web service descriptor and click **OK** to continue with refresh the web service descriptor.
4. If the **Source URI** property specifies a location for the original WSDL document, Designer asks you if you want to use a different source file for refreshing the web service descriptor. Click **Yes** to select a file at a new location as the source. Click **No** to use the file at the specified location as the source.

   If you selected **No** to use the specified location and Designer cannot read the WSDL file at that location, Designer displays a message prompting you to cancel the refresh or to select a new source location. Click **Cancel** to cancel the refresh. Click **OK** to specify a new location.
5. If you indicated that you wanted to select a new source location in the previous step, or if the Source URI property has no value, in the Refresh Web Service Descriptor dialog box, select the location of the WSDL file do one of the following:

Select...	To refresh a web service descriptor using...
CentraSite	A service asset in CentraSite
File/URL	A WSDL document that resides on the file system or on the Internet.
UDDI	A WSDL document in a UDDI registry

6. Click **Next**.

7. If you selected **CentraSite** as the source, under **Select Web Service from CentraSite**, select the service asset in CentraSite that you want to use to create the web service descriptor. Click **Next**.

   Designer filters the contents of the Services folder to display only service assets that are web services.

   If Designer is not configured to connect to CentraSite, Designer displays the **CentraSite > Connections** preference page and prompts you to configure a connection to CentraSite.

8. If you selected **File/URL** as the source, do one of the following:

   - Enter the URL for the WSDL document. The URL should begin with http:// or https://.
   - Click **Browse** to navigate to and select a WSDL document on your local file system.

9. If you selected **UDDI** as the source, under **Select Web Service from UDDI Registry**, select the web service from the UDDI registry. Click **Next**.

   If Designer is not currently connected to a UDDI registry, the Open UDDI Registry Session dialog box appears. Enter the details to connect to the UDDI registry and click **Finish**.

10. Click **Next** if you want to specify different prefixes than those specified in the XML schema definition. If you do not want to change the prefixes specified in the XML schema definition, click **Finish**.

11. On the Assign Prefixes panel, if you want the web service descriptor to use different prefixes than those specified in the XML schema definition or modified at the time of creating the web service descriptor, select the prefix you want to change and enter a new prefix. Repeat this step for each namespace prefix that you want to change.

    Note:
    The prefix you assign must be unique and must be a valid XML NCName as defined by the specification http://www.w3.org/TR/REC-xml-names/#NT-NCName.

12. Click **Finish**.

Designer refreshes the web service descriptor. If Designer cannot refresh a web service descriptor, Designer rolls back to the last saved version of the web service descriptor. If refresh is not successful, use the messages returned by Designer and the messages in the error log to determine why.

Viewing the WSDL Document for a Web Service Descriptor

On the WSDL tab, you can view the WSDL document associated with a consumer or provider web service descriptor.

- For a consumer web service descriptor, the WSDL document is a local copy of the original WSDL document used to create the consumer web service descriptor with the following changes:
  - The addition of any headers or faults added to the consumer web service descriptor.
  - Modifications to the editable properties of the provider web service descriptor or its constituents, such as the use of a web service endpoint alias.
- For a service first provider web service descriptor, the WSDL document contains all the information a consumer needs to create a web service client that invokes the operations described in the WSDL.
- For a WSDL first provider web service descriptor, the WSDL document is the original source WSDL with the following changes:
  - Addition of any headers or faults added to the provider web service descriptor.
  - Modifications to the editable properties of the provider web service descriptor or its constituents, such as the use of a web service endpoint alias.
  - Modification to the `name` attribute in the `wsdl:service` element to reflect the name of the web service descriptor.
  - Removed all `soapjms` wsdl extensions for JMS bindings contained in the original `wsdl:port`, `wsdl:service`, or `wsdl:binding` elements.
  - Addition of `soapjms` wsdl extensions to the `wsdl:binding` element for JMS bindings. This includes JMS transport properties defined in the web service endpoint alias assigned to the binder that specifies the JMS transport.
  - Changes to the `location` attribute of the `wsdl:port` element to reflect any JMS connection related settings for the JMS URI.
- The displayed WSDL document contains all the information a consumer needs to create a web service client that invokes the operations described in the WSDL.
- For a web service descriptor created from a WSDL that contains relative URIs that are anonymously addressable, Integration Server replaces any relative URIs with an absolute URI using the base URI of the WSDL file.
- If you attach a WS-Policy to a provider web service descriptor that is *not* running in pre-8.2 compatibility mode (i.e., the **Pre-8.2 compatibility mode** property is set to false), the generated

WSDL will be annotated with the policy. If you attach multiple policies to a web service descriptor, the generated WSDL will have policy annotations of all the attached policies. The policy is annotated using `PolicyURIs` attributes. Integration Server identifies the associated policies by specifying the policy IDs in the `PolicyURIs` attributes.

You must save the web service descriptor before the policy annotations will be included in the WSDL. If the web service descriptor is not saved after a policy is attached to it, the WSDL on the WSDL tab will not yet include the policy annotations. When you save the web service descriptor, Integration Server obtains the policy from the policy files so that Designer can display it in the generated WSDL.

- When viewing the WSDL for a WSDL first provider web service descriptor that was created from a policy annotated WSDL, the generated WSDL will be annotated with the *attached* policies. The generated WSDL will not include the annotated policy from which it was generated.
- For a consumer web service descriptor, the generated WSDL will always contain the original annotated policy from the source WSDL document.

To view the WSDL document for a web service descriptor

1. In Package Navigator view, open the web service descriptor for which you want to view the WSDL document.
2. Click the WSDL tab.

   Designer displays the WSDL document for the web service descriptor.

WS-I Compliance for Web Service Descriptors

The WS-I option specifies whether the web service descriptor enforces compliance with the *WS-I Basic Profile 1.1*, a set of guidelines for using web services specifications to maximize interoperability (including guidance for such core web services specifications such as SOAP, WSDL, and UDDI).

As an example, using the RPC/Encoded style and use is not supported by the WS-I profile. If a web service descriptor makes use of the RPC/Encoded style, and **WS-I compliance** is enabled, Designer displays indicating that the WSD is not compliant and prompts you to save the WSD as non-compliant.

Enforcing WS-I compliance also affects the contents and signature for operations in the WSD. For example, the use of multiple top-level fields is not supported in the WS-I profile; if a service (operation) in a provider WSD includes multiple top-level fields, Designer prompts you to save the WSD as non-compliant

The WS-I compliance option is set to **Yes** or **No** when you create a web service descriptor (No is the default). You can modify this option by changing the **WS-I compliance** property.

Note:
The WS-I profile only address the SOAP 1.1 protocol.

Modifying WS-I Compliance for a Web Service Descriptor

The WS-I compliance option is set to **Yes** or **No** when you create a web service descriptor (**No** is the default). You can modify this option by changing the **WS-I compliance** property. Keep the following points in mind when determining whether to enforce WS-I compliance:

- The WS-I profiles only address the SOAP 1.1 protocol. If the web service descriptor is using the SOAP 1.2 protocol, Designer will display an error message when **True** is selected
- WS-I Basic Profile 1.0 supports only HTTP or HTTPS bindings. Consequently, WS-I compliance cannot be enforced if the WSDL contains a SOAP over JMS binding. The **WS-I compliance** property cannot be set to true if a web service descriptor has a JMS binder.

To modify WS-I compliance for a web service descriptor

1. In Package Navigator view, open and lock the web service descriptor for which you want to change WS-I compliance enforcement.
2. In the Properties view, next to **WS-I compliance**, select **True** if you want Integration Server to enforce WS-I Basic Profile 1.1 compliance. Otherwise, select **False**.
3. Click **File > Save**.

Reporting the WS-I Profile Conformance for a Web Service Descriptor

You can analyze a web service descriptor for conformance to the WS-I Basic Profile 1.1.

To analyze whether the web service descriptor is WS-I Compliant, you must be connected to the Internet. To enable connecting to the Internet, ensure that you provide the appropriate proxy server settings in **Window > Preferences > General > Network Connections**. For more information about setting the proxy server details, see *Software AG Designer Online Help*.

To analyze a web service descriptor for WS-I Profile Conformance

1. In Package Navigator view, open and lock the web service descriptor that you want to analyze for WS-I Profile Compliance.
2. Click to check whether the web service descriptor is WS-I Profile Compliant.

   If the web service descriptor is WS-I Compliant, Designer displays a confirmation message.

   If the web service descriptor is not WS-I Compliant, Designer displays the error details in the Problems view.

Changing the Target Namespace for a Web Service Descriptor

For a service first provider web service descriptor, you can change the target namespace that Integration Server uses in the generated WSDL document. The target namespace specifies the XML namespace to which the elements, attributes, and type definitions in the WSDL belong.

You can only change the target namespace for service first provider WSD. For a WSDL first provider WSD or a consumer WSD, the target namespace is determined by the WSDL document used as the source.

To change the target namespace for a service first provider web service descriptor

1. In Package Navigator view, open and lock the service first provider WSD for which you want to change the target namespace.

2. In the Properties view, in the **Target namespace** field, specify the URL that you want to use as the target namespace for elements, attributes, and type definitions in the WSDL generated for this provider WSD.

3. Click **File > Save**.

Viewing the Namespaces Used within a WSDL Document

You can view a list of all the XML namespaces used by the web service descriptor when it is first created. You can also view the prefix associated with each XML namespace.

To view the namespaces and prefixes used in a web service descriptor

1. In Package Navigator view, open the service first provider WSD for which you want to view the list of XML namespaces used in the original WSDL document.

2. In the Properties view, click the browse button in the **Namespaces** field.

   The Namespaces dialog box appears displaying a list of XML namespaces used within the WSDL document. This is an array of namespace prefixes and their associated XML namespace names. This information is not editable.

3. Click **OK** to close the dialog box.

Enabling MTOM/XOP Support for a Web Service Descriptor

The Message Transmission Optimization Mechanism (MTOM) feature provides optimization of binary message transportation using XOP (XML-binary Optimized Packaging). If attachments are enabled for the web service descriptor, instances of XML-type base64Binary are transported using MIME attachments, which improves the performance of large binary payload transport. Integration

Server supports SOAP attachments only for web service descriptors that specify style/use of RPC/Literal or Document/Literal.

Integration Server supports streaming the SOAP attachments based on the MTOM/XOP standards for both inbound and outbound messages. For more information about the configuration required to enable MTOM streaming, see the *Web Services Developer's Guide*.

Enabling SOAP Attachments for a Web Service Descriptor

Before enabling SOAP attachments for a web service descriptor, configure the watt.server.SOAP.MTOMThreshold property with the appropriate value. This property specifies the field size, in kilobytes, that determines whether Integration Server sends base64binary encoded data in an outbound SOAP message as a MIME attachment or whether it sends it inline in the SOAP message. For more information about this property, see *webMethods Integration Server Administrator's Guide*.

If you want to stream MTOM attachments, you have to do additional configuration. For more information about the configuration required to enable MTOM streaming, see the *Web Services Developer's Guide*.

To enable SOAP attachments for a web service descriptor

1. In Package Navigator view, open and lock the web service descriptor for which you want to enable or disable SOAP attachments.

2. In the Properties view, next to **Attachment enabled**, select **True** if you want to enable SOAP attachments for the WSD. Otherwise, select **False**.

3. Click **File > Save**.

Using pub.string:base64Encode with MTOM Implementations

By default, the public service pub.string:base64Encode inserts a new line after 76 characters of data. This is not the canonical lexical form expected by MTOM implementations. If you use this public service rather than a custom service for base64 encoding, you can use the optional input parameter *useNewLine* to remove the line break and the optional input parameter *encoding* to change the encoding. The default value for *useNewLine* is true, which keeps the line break at 76 characters. If you do not specify the encoding, ASCII will be used.

If you use the public service pub.string:base64Decode for base64 decoding, you can use the optional input parameter encoding to change the encoding. If you do not specify the *encoding*, ASCII will be used.

Adding SOAP Headers to the Pipeline

For a web service descriptor, you can instruct Integration Server to add the contents of SOAP headers to the pipeline, making the contents of the SOAP headers available to subsequent services.

The value of the **Pipeline headers enabled** property determines whether or not Integration Server places the contents of the SOAP header in the pipeline as a document named *soapHeaders*. The default value is false.

If the **Pipeline headers enabled** property is set to true for a provider WSD, when an IS service that corresponds to an operation in the WSD is invoked, Integration Server places the contents of the SOAP request header in the input pipeline for the IS service.

If the **Pipeline headers enabled** property is set to true for a consumer WSD, when one of the web service connectors is invoked, Integration Server places the contents of the SOAP response header in the output pipeline for the web service connector.

For detailed information about the content and structure of the *soapHeaders* document that Integration Server adds to the pipeline, see *Web Services Developer's Guide*.

Note:
For web service descriptors contained in packages created in versions of Integration Server prior to 8.0, the **Pipeline headers enabled** property is set to **True**.

To add SOAP headers to the pipeline

1. In Package Navigator view, open and lock the web service descriptor for which you want to enable or disable adding SOAP headers to the pipeline.

2. In the Properties view, next to **Pipeline headers enabled**, select **True** if you want to enable SOAP headers for the web service descriptor. Otherwise, select **False**.

3. Click **File > Save**.

Validating SOAP Response

For a consumer web service descriptor, you can indicate whether or not Integration Server validates a SOAP response received by any web service connectors within the consumer WSD.

The value of the watt.server.SOAP.validateResponse server configuration parameter determines whether or not the **Validate SOAP response** property is honored or ignored.

- When watt.server.SOAP.validateResponse is set to **true**, the value of the **Validate SOAP response** property determines whether or not Integration Server validates the SOAP response.
- When watt.server.SOAP.validateResponse is set to **false** (or anything besides **true**), Integration Server ignores the **Validate SOAP response** property and does not validate the SOAP response.

By default, the watt.server.SOAP.validateResponse is set to **true**.

To validate SOAP responses received by web service connectors

1. In Package Navigator view, open and lock the consumer WSD for which you want to enable or disable SOAP response validation.

2. In the Properties view, next to **Validate SOAP response**, select **True** if you want Integration Server to validate SOAP response messages received by web service connectors included with this consumer WSD. Otherwise, select **False**.

3. Click **File > Save**.

Validating Schemas Associated with a Web Service Descriptor

To help ensure interoperability between a web service descriptor and other web service vendors or clients, you can use Integration Server to validate the schemas associated with the web service descriptor. Integration Server provides an internal schema parser that it uses to validate XML schema definitions at the following times:

- When you create or refresh a consumer web service descriptor or WSDL first provider web service descriptor from a WSDL document.
- When you change the IS schemas, document types, or signatures of the services associated with a web service descriptor.

While Integration Server uses an internal schema parser to validate the schemas automatically, you can also instruct Integration Server to use the Xerces Java parser. The Xerces Java parser provides stricter validation than that provided by the Integration Server internal schema parser. As a result, some schemas that the internal schema parser considers to be valid might be considered invalid by the Xerces Java parser.

Integration Server uses the Xerces Java parser to validate the schemas associated with a web service descriptor at the following times:

- When you create a consumer web service descriptor or WSDL first provider web service descriptor from a WSDL document. In the New Web Service Descriptor wizard, Designer provides an option named **Validate Schema using Xerces**. When selected, Integration Server validates the schemas defined or referenced in the WSDL document. If the Xerces Java parser determines the schema(s) are invalid, Integration Server does not create the web service descriptor and Designer displays the validation errors.
- When you refresh a consumer web service descriptor or WSDL first provider web service descriptor for which the **Validate Schema using Xerces** was selected at the time the web service descriptor was created.
- When you create a service first provider web service descriptor. In the New Web Service Descriptor wizard, Designer provides an option named **Validate Schema using Xerces**. When selected, as part of creating a service first provider web service descriptor, Integration Server converts the signatures of the services used as operations to XML schema elements. Then Integration Server uses the Xerces Java parser to validate the schema elements. If the Xerces Java parser determines the schema(s) are invalid, Integration Server does not create the web service descriptor and Designer displays the validation errors.
- When the **Validate Schema using Xerces** property is set to true for a web service descriptor and one of the following occurs:

- You change the IS schemas, document types, or signatures of the services associated with a web service descriptor.
- You select an element declaration from an XML Schema definition to use as the input or output signature of a 6.5 SOAP-MSG style operation. For more information about using 6.5 SOAP-MSG style services as operations, see "Using a 6.5 SOAP-MSG Style Service as an Operation" on page 802.
- You refresh the web service connectors for a consumer web service descriptor.

While validation by the Xerces Java parser can increase the time it takes to create, update, or refresh a web service descriptor and increase the time to refresh or update a web service connector, using stricter validation can help ensure interoperability with other web service vendors.

Note: Integration Server uses Xerces Java parser version J-2.11.0. Limitations for this version are listed at http://xerces.apache.org/xerces2-j/xml-schema.html.

When validating XML schema definitions, Integration Server uses the Perl5 regular expression compiler instead of the XML regular expression syntax defined by the World Wide Web Consortium for the XML Schema standard. As a result, in XML schema definitions consumed by Integration Server, the pattern constraining facet must use valid Perl regular expression syntax. If the supplied pattern does not use proper Perl regular expression syntax, Integration Server considers the pattern to be invalid.

Note:
If the watt.core.datatype.usejavaregex configuration parameter is set to true, Integration Server uses the Java regular expression compiler instead of the Perl5 regular expression compiler. When the parameter is true, the pattern constraining facet in XML schema definitions must use valid syntax as defined by the Java regular expression.

Enabling Xerces Schema Validation for a Web Service Descriptor

When you create a web service descriptor from a WSDL document, you can specify that Integration Server use the Xerces Java parser to validate the schemas associated with the WSDL document. Once the web service descriptor exists, you can indicate that Integration Server validate the schemas associated with the web service descriptor by setting the **Validate Schema using Xerces** property to true. When the **Validate Schema using Xerces** property is set to true, Integration Server validates the schemas associated with an existing web service descriptor in the following situations:

- You change the IS schemas, document types, or signatures of the services associated with a web service descriptor

  Note: Integration Server uses the internal schema processor to validate the schemas at this point as well.

- You select an element declaration from an XML Schema definition to use as the input or output signature of a 6.5 SOAP-MSG style operation. For more information about using 6.5 SOAP-MSG style services as operations, see "Using a 6.5 SOAP-MSG Style Service as an Operation" on page 802.

- You refresh the web service connectors for a consumer web service descriptor.

Integration Server sets the **Validate Schema using Xerces** property to true for all new web service descriptors. If you migrated a web service descriptor from a previous version of Integration Server, the migration utility set the value based on the version of Integration Server from which the web service descriptor was migrated.

- If the web service descriptor was migrated from Integration Server version 7.1.x, the migration utility set the **Validate Schema using Xerces** property to true.
- If the web service descriptor was migrated from Integration Server version 8.x, the migration utility used the value of the watt.server.wsdl.validateWSDLSchemaUsingXerces parameter to determine the value of the **Validate Schema using Xerces** property. If the parameter was set to true, the migration utility set the property to true. It the parameter was set to false, the migration utility set the property to false.

  Note:
  The watt.server.wsdl.validateWSDLSchemaUsingXerces parameter was removed in Integration Server version 9.0.

To enable or disable schema validation by the Xerces Java parser

1. In Package Navigator view, open and lock the web service descriptor for which you want to enable or disable schema validation by the Xerces Java parser.

2. In the Properties view, next to **Validate schema using Xerces**, select **True** if you want Integration Server to use the Xerces Java parser to validate the XML Schema definitions associated with the web service descriptor. Otherwise, select **False**. The default is **True**.

3. Click **File > Save**.

Omitting xsd:any from the WSDL Document

When generating the WSDL schema for a provider web service descriptor that contains a document reference with the **Allow unspecified fields** set to **True**, Integration Server uses the value of the watt.core.schema.createSchema.omitXSDAny server configuration parameter to determine whether the xsd:any element is omitted from or retained in the schema.

The value of watt.core.schema.createSchema.omitXSDAny is global to all web service descriptors. However, you can override the value of the server configuration parameter for an individual web service descriptor by using the **Omit xsd:any from WSDL** property. If the value of this property is true, the xsd:any element is omitted from the schema portion of the WSDL document whether or not the document variable has **Allow unspecified fields** set to **True**. If the value is false, the xsd:any element is retained in the schema portion *only if* the document variable has **Allow unspecified fields** set to **True**.

To omit or retain xsd:any in the WSDL document

1. In Package Navigator view, open and lock the provider web service descriptor for which you want to omit or retain the xsd:any element when generating the WSDL schema.

2. In the Properties view, next to **Omit xsd:any from WSDL**, select **True** if you want Integration Server to omit the xsd:any element from the WSDL document associated with the web service descriptor. Otherwise, select **False**.

   The default is **True**.

3. To save the changes, click **File > Save**.

4. For the changes to take effect, right-click the web service descriptor and select **Refresh Web Service Descriptor**.

   Alternatively, you can reload the package that contains the web service descriptor. For more information, see *webMethods Service Development Help*.

Working with Binders

A *binder* is a webMethods term for a collection of related definitions and specifications for a particular port. The binder is a container for the endpoint address, WSDL binding element, transport protocol, and communication protocol information. Designer creates at least one binder when it generates the web service descriptor based on the data in the WSDL or IS service. The Binders tab displays the binders defined for a web service descriptor.

You can add new binder definitions to a service first provider web service descriptor. Binders cannot be added to a WSDL first provider web service descriptor or a consumer web service descriptor.

You can define a separate binder for each combination of endpoint address and protocol information that you want the service first provider web service descriptor to support.

Binders and Mixed Use

Integration Server and Designer do not support mixed “use” across binders and operations in a single web service descriptor. That is, the binders in a web service descriptor must specify the same value for the **SOAP binding use** property.

Integration Server and Designer enforce this restriction in the following way:

- In a service first provider web service descriptor, the first binder determines the “use” for all subsequent binders. If the first binder specifies a **SOAP binding use** of “literal”, any additional binder added to the provider web service descriptor must specify literal as the **SOAP binding use**.
- When creating a WSDL first provider web service descriptor or a consumer web service descriptor from a WSDL document, Integration Server will not create the web service descriptor if the WSDL document contains bindings with different use value or operations with different use values. Integration Server throws the following exception:

```
[ISS.0085.9285] Bindings or operations with mixed "use" are not
supported.
```

Existing Web Service Descriptors with Mixed Use Binders

Integration Server continues to support existing web service descriptors that contain binders with mixed use as long as the web service descriptors are not modified. Once the web service descriptor is modified, Designer will not save the web service descriptor if it has mixed use binders.

To edit and save an existing provider web service descriptor with mixed binders, create separate provider web service descriptors for each binder use. For example, if a provider web service descriptor contains binder1 which specifies a “use” of literal and binder2 which specifies a “use” of encoded, copy the provider web service descriptor. In the provider web service descriptor copy, remove binder1. In the original provider web service descriptor, remove binder2. The provider web service descriptors can then be saved.

Binders and Mixed Style

Integration Server and Designer do not support mixed “style” across binders in a single web service descriptor if the web service descriptor does not run in pre-8.2 compatibility mode. That is, the binders in a web service descriptor for which the **Pre-8.2 compatibility mode** property is set to false must specify the same value for the **SOAP binding style** property.

Integration Server and Designer enforce this restriction in the following way:

- In a service first provider web service descriptor, the first binder determines the “style” for all subsequent binders. If the first binder specifies a **SOAP binding style** of “document”, any additional binder added to the provider web service descriptor must specify document as the **SOAP binding style**.
- When creating a WSDL first provider web service descriptor from a WSDL document, Integration Server will not create the web service descriptor if the services reference bindings with different styles.
- When creating a consumer web service descriptor from a WSDL document, Integration Server will not create the web service descriptor if the services reference supported bindings that specify mixed style values.

Note:
The restriction on mixed binding styles across binders does not apply to web service descriptors that run in pre-8.2 compatibility mode.

Adding a Binder to Web Service Descriptor

Keep the following points in mind when creating a new binder:

- You can add a binder definition to a service first provider web service descriptor.
- All existing operations will be duplicated within the new binder.

- For a web service descriptor that runs in pre-8.2 compatibility mode (**Pre-8.2 compatibility mode** property is set to true), the new binder must specify the same "use" as the binder that already exists in the provider web service descriptor. For more information about mixed use in binders, see "Binders and Mixed Use" on page 790.
- For a web service descriptor that does not run in pre-8.2 compatibility mode (**Pre-8.2 compatibility mode** property is set to false), the new binder must specify the same "style" *and* "use" as the binder that already exists in the provider web service descriptor. For more information about mixed styles in binders, see "Binders and Mixed Style" on page 791.
- You can add a binder that specifies the JMS transport only if a valid provider web service endpoint alias exists for the JMS transport. For example, if the only web service endpoint alias that exists for JMS specifies a SOAP-JMS trigger that no longer exists, Integration Server does not consider the endpoint alias to be valid. Consequently, the endpoint alias cannot be assigned to a JMS binder. For more information about creating a web service endpoint alias, see *webMethods Integration Server Administrator's Guide*
- You can only add a JMS binder to a web service descriptor that does not run in pre-8.2 compatibility mode (**Pre-8.2 compatibility mode** property is set to false).
- In a JMS binder for a provider web service descriptor, the property values under JMS Settings and JMS Message Details are set by the web service endpoint alias assigned to the binder. The JMS Settings and JMS Message Details properties are read-only.
- If the **WS-I compliance** property is set to **True**, you can only create binders that comply with the WS-I profile.

To add a binder to a service first provider web service descriptor

1. In the Package Navigator view in the Service Development perspective, open and lock the provider web service descriptor to which you want to add a binder.
2. In the Binders tab, click on the web service descriptor toolbar or right-click and select **Add Binder**.
3. In the New Binder Options dialog box, specify the following information:

In this field...	Specify...
SOAP Version	Whether SOAP messages for this web service should use **SOAP 1.1** or **SOAP 1.2** message format.
Transport	The transport protocol used to access the web service. Select one of the following: ■ **HTTP** ■ **HTTPS** ■ **JMS**

In this field...	Specify...
Use and Style for Operations	The style/use for operations in the provider web service descriptor. Select one of the following: ■ **Document - Literal** ■ **RPC - Literal** ■ **RPC - Encoded**
Endpoint	The address at which the web service can be invoked. Do one of the following: ■ To use a provider web service endpoint alias to specify the address, select the **Alias** option. Then, in the **Alias** list, select the provider web service endpoint alias. Select DEFAULT(*aliasName*) if you want to use the information in the default provider web service endpoint alias for the address. If the **Alias** list includes a blank row, the Integration Server does not have a default provider web service endpoint alias for the protocol. **Note:** If a default provider endpoint alias is later set for the selected protocol, then Integration Server uses the information from the alias when constructing the WSDL document and during run-time processing. ■ To specify a host and port as the address, select the **Host** option. Then, in the **Host** field specify the host name for the Integration Server on which the web service resides. In the **Port** field, specify an active HTTP or HTTPS listener port defined on the Integration Server specified in the **Host** field. **Note:** You can only specify **Host** and **Port** for the endpoint if a default provider endpoint alias does not exist for the selected protocol. When a default alias exists, Designer populates the **Host** and **Port** fields with the host and port from the default provider end point alias. **Note:** If you selected JMS as the transport, you must specify an alias. After you select a provider web service endpoint alias, Designer displays the initial portion of the JMS URI that will be used as the address in the **Port address (prefix)** field.
Directive	The SOAP processor used to process the SOAP messages received by the operation in the provider web service descriptor. The Directive list

In this field...	Specify...
	displays all of the SOAP processors registered on the Integration Server. The default processor is **ws - Web Services SOAP Processor**.

4. Click **OK**. Designer adds the new binder to the Binders tab.

5. Click **File > Save**.

Notes:

- If you specify HTTP or HTTPS as the transport, but do not specify a host, port, or provider web service endpoint alias and there is not a default provider endpoint alias for the transport protocol, Integration Server uses the primary port as the port in the endpoint URL. If the selected transport and the protocol of the primary port do not match, web service clients will not execute successfully. For more information see "Protocol Mismatch Between Transport and Primary Port" on page 752.
- You can change the default name that Designer assigns to the binder. You can rename the binder by changing the value of the **Binder name** property or by selecting the new binder, right-clicking it, and selecting **Rename**.

Copying Binders Across Provider Web Service Descriptors

You can cut or copy an existing binder from another provider web service descriptor and paste it into the Binders tab of a service first provider web service descriptor. (Note that drag and drop of binders is not supported.)

Keep the following points in mind when pasting in a binder from another provider web service descriptor:

- The endpoint, directive, WS-I, transport, and use-style values are the same as those in the original (source) binder. You can modify these in the Properties view.
- All operations in the cut or copied binder are carried with it. If a pasted binder contains an operation that is not already in the web service descriptor, the operation is added to the web service descriptor.

Changing the Binder Transport

You can change the specified transport of a binder in a service first provider web service descriptor by changing the value of the binder **Transport** property. Keep the following points in mind when changing the binder transport:

- A binder can specify the JMS transport only if the web service descriptor does not run in compatibility mode (the **Pre-8.2 compatibility mode** property is set to false).
- You can change the transport to JMS only if a provider web service endpoint alias that specifies the JMS transport exists already.

- If you change the transport from HTTP or HTTPS to JMS, Designer automatically assigns the first valid provider web service endpoint alias that specifies the JMS transport to the **Port alias** property of the binder. If there is not valid endpoint alias for JMS, the binder transport cannot be changed. For example, if the only web service endpoint alias that exists for JMS specifies a SOAP-JMS trigger that no longer exists, Integration Server does not consider the endpoint alias to be valid. Consequently, the endpoint alias cannot be assigned to a JMS binder.

  Additionally, Designer updates the **Port address** property to display the initial part of the JMS URI, specifically "jms":<lookup var>:<dest>?targetService.

- If you change the transport from JMS to HTTP or HTTPS, Designer deletes the values of the **Port alias** property and the **Port address** property. If the Integration Server identifies a default provider endpoint alias for the protocol used by the binder, Designer sets **Port alias** property to DEFAULT(*aliasName*). If you want to use a specific web service endpoint alias to specify the hostname and port for the provider web service endpoint URL, make sure to specify that alias in the **Port alias** property.

- When you change the transport, Designer updates the **Binding type** and **SOAP binding transport** properties to match the selected transport.

Deleting a Binder from a Web Service Descriptor

You can delete a binder from a service first provider web service descriptor.

A web service descriptor must contain at least one binder. If you delete the last binder in a web service descriptor, you must add a new binder before the web service descriptor can be saved (the binder can be empty).

To delete an operation from a binder in a provider web service descriptor

1. In the Package Navigator view in theService Development perspective, open and lock the service first provider web service descriptor from which you want to delete a binder.

2. In the Binders tab, select the binder to delete.

3. Click on the web service descriptor toolbar or right-click and select **Delete**.

4. Click **File > Save**.

Deleting an Operation from a Binder

If two binders share an operation, you can delete the operation from one binder but not the other. The operation will still be in the provider web service descriptor, but only be in one binder.

Keep the following in mind when deleting an operation from a binder:

- You can delete operations from a service first provider web service descriptor only.

- If other binders in the provider web service descriptor contain the operation, that operation remains in the web service descriptor.
- If an operation only exists in one binder, deleting it from that binder removes it entirely from the web service descriptor.
- If you delete an operation from the Operations tab, it is deleted entirely from the web service descriptor and from all the binders that exist for that web service descriptor. For more information about deleting operations, see "Deleting Operations" on page 804.

To delete an operation from a binder in a provider web service descriptor

1. In the Package Navigator view in the Service Development perspective, open and lock the service first provider web service descriptor.
2. In the Binders tab, expand the binder containing the operation to delete.
3. In the binder, select the operations you want to delete.
4. Click on the web service descriptor toolbar or right-click and select **Delete**.

   Designer deletes the selected operation from the binder.
5. Click **File > Save**.

Modifying the SOAP Action for an Operation in a Binder

For a service first provider web service descriptor, you can change the SOAP action specified for an operation in a binder. By default, the SOAP action uses the format *binderName_operationName* to ensure that it is unique within a web service. At run time, the values associated with a SOAP action are used to find the actual operation being invoked. For more information about how Integration Server determines which operation to invoke, see *Web Services Developer's Guide*.

To modify the SOAP action property for an operation in a binder

1. In the Package Navigator view in the Service Developmentperspective, open and lock the provider web service descriptor.
2. In the Binders tab, select the binder containing the operation for which you want to edit the SOAP action.
3. In the Properties view, next to the **SOAP action** property, click the browse button. Designer displays the SOAP Action dialog box which identifies the SOAP action string associated with each operation in the selected binder.

4. For the operation whose SOAP action you want to change, enter the new SOAP action value in the **SOAP Action** column. Make sure that the new SOAP Action value is unique across the web service descriptor.

5. Click **OK**.

   Designer applies the SOAP action change to the operation in this binder only.

6. Click **File > Save**.

Assigning a Web Service Endpoint Alias to a Binder

You can associate a web service endpoint alias with a binder in a provider or consumer web service descriptor. A web service endpoint alias represents the network address and, optionally, any security credentials to be used with web services. The network address properties can be used to enable dynamic addressing for web services. The security credentials can be used to control both transport-level and message-level security for web services.

For a consumer web service descriptor and its associated web service connectors (WSC), the alias information (including the addressing information and any security credentials), is used at run time to generate a request and invoke an operation of the web service. For web service connectors behind a firewall, the endpoint alias also specifies the proxy alias for the proxy server through which Integration Server routes the web service request. For more information about proxy server usage, see *webMethods Integration Server Administrator's Guide*.

For a provider web service descriptor, the endpoint alias is used to construct the "location=" attribute of the soap:address element within the wsdl:port element when WSDL is requested for the web service. The security credentials may be used when constructing a response to a web service request.

For information about creating a web service endpoint alias, see *webMethods Integration Server Administrator's Guide*.

To assign a web service endpoint alias to a binder

1. In the Package Navigator view in the Service Development perspective, open and lock the web service descriptor to which you want to associate the web service endpoint alias.

2. In the Binders tab, select the binder to which you want to assign an endpoint alias.

3. In the Properties view, next to the **Port alias** property, select the web service endpoint alias that you want to associate with the web service descriptor. Designer lists only those endpoint aliases of the same type as the web service descriptor and whose protocol matches the binder protocol. If there have been changes to the web service endpoint aliases since you connected Designer to Integration Server, use Designer to refresh the connection to Integration Server.

   If this is a provider web service and the binder protocol is HTTP or HTTPS, you can assign the default provider endpoint alias to the binder. Select DEFAULT(*aliasName*) if you want to

use the information in the default provider web service endpoint alias for the address. If the **Alias** list includes a blank row, Integration Server does not have a default provider web service endpoint alias for the protocol.

Note:
If you select the blank row and a default provider endpoint alias is later set for the selected protocol, Integration Server then uses the information from the alias when constructing the WSDL document and during run-time processing.

4. Click **File > Save**.

Notes:

- When the **Port alias** property is modified for a consumer web service descriptor and the web service descriptor is viewed on the WSDL tab, the generated WSDL does not reflect the change to the port alias. However, the new value will be used at run-time.
- After assigning an alias to a JMS binder in a provider web service descriptor, if the web service endpoint alias specifies a SOAP-JMS trigger, the web service descriptor has a dependency on the SOAP-JMS trigger. Consequently, at start up or when reloading the package containing the web service descriptor, Integration Server must load the SOAP-JMS trigger before loading the web service descriptor. If the SOAP-JMS trigger and web service descriptor are not in the same package, you need to create a package dependency. The package that contains the web service descriptor must have a dependency on the package that contains the SOAP-JMS trigger.

Configuring Use of the Client Side Queue

You can enable use of the client side queue for a JMS binder in a consumer web service descriptor. The client side queue is a message store that contains JMS messages sent during service execution when the JMS provider was not available. Each JMS connection alias has its own client side queue. When the JMS provider becomes available, Integration Server sends messages from the client side queue to the JMS provider.

When use of the client side queue is enabled for a JMS binder and the JMS provider is not available at the time a web service connector sends a message using the JMS binding, Integration Server writes the message to the client side queue.

When use of the client side queue is disabled for a JMS binder and the JMS provider is not available at the time the web service connector executes, Integration Server throws an ISRuntimeException. Integration Server includes the exception in the *fault* document returned to the web service connector.

Keep the following points in mind when enabling use of the client side queue for a JMS binder:

- The client side queue associated with the JMS binder is determined by the JMS connection alias in the consumer web service endpoint alias for the binder. The maximum size of the client side queue must be greater than zero. If the JMS connection alias sets the size of the client side queue to zero (**Maximum Queue Size** is set to 0), the client side queue is effectively disabled. Integration Server will not write messages to a client side queue that has a maximum size of

0 messages. For more information about configuring a JMS connection alias, see *webMethods Integration Server Administrator's Guide*

- The client side queue can be used with web service connectors for In-Only and In-Out operations. For an In-Out operation, the reply to destination for the web service must be a non-temporary queue.

To configure the use of the client side queue for a JMS binder

1. In the Package Navigator view in theService Development perspective, open and lock the web service descriptor containing the binder for which you want to configure the use of the client side queue.

2. In the Binders tab, select the JMS binder for which you want to configure the use of the client side queue.

3. In the Properties view, next to the **Use CSQ** property, select **True** to enable use of the client side queue. If you do not want Integration Server to use the client side queue for JMS messages sent using the binding represented by this binder, select **False**.

4. Click **File > Save**.

Working with Operations

An *operation* is the WSDL element that exposes some functions of a web service and defines how data is passed back and forth. In a web service descriptor, an operation corresponds to a service on Integration Server.

Each operation contains a single request and a single response. Each request and response contains a single, read-only body element and one or more header elements. A response can also contain fault elements.

The *body elements* contain the application-defined XML data being exchanged in the SOAP message:

- In a service first provider WSD, the body elements represent the signature of the service. The body element in the request contains the input properties. The body element in the response contains the output properties.
- In a WSDL first provider WSD or a consumer WSD, the input/output properties and the body element are defined by the remote WSDL document. Neither the input/output definitions nor the operations can be changed, added, or deleted.

A *header element* defines the format of the SOAP headers that may be present in a SOAP message (request or response). Headers are optional and can be added to or deleted from any web service descriptor.

A *fault element* provides a definition for a SOAP fault (that is, the response returned to the sender when an error occurs while processing the SOAP message). Fault elements are optional and can be added to or deleted from any web service descriptor.

Adding Operations

When you add operations to a service first provider WSD, the operations are also added to every binder in the WSD. The values defined by a specific binder will apply to the operation.

Note:
You can add operations to a service first provider WSD only.

You can add operations by:

- Adding one or more IS services from the Package Navigator. Each service will be converted to an operation in the provider WSD.
- Copying or moving an operation from another provider WSD.
- Adding a 6.5 SOAP-MSG Style Service as an operation.

Adding an IS Service as an Operation

Keep the following points in mind when adding an IS service as an operation to an provider web service descriptor:

- You can add operations to a service first provider WSD only.
- A 6.5 SOAP-MSG style service can only be added as an operation if it meets the requirements identified in "Using a 6.5 SOAP-MSG Style Service as an Operation" on page 802.
- Because Integration Server and Designer do not support mixed "use" across binders and operations and mixed "style" across binders in a single web service descriptor, the service signature must meet the style/use signature requirements established by the existing binder. For more information, see "Service Signature Requirements for Service First Provider Web Service Descriptors" on page 746.

To add an IS service to a service first provider web service descriptor

1. In Package Navigator view, open and lock the service first provider WSD to which you want to add an IS service as an operation.

2. On the web service descriptor editor toolbar, click or right-click in the Operations tab and select **Add Operation**.

3. In the Select one or more services to include in the web service descriptor dialog box, select one or more services and click **OK**.

   The specified operations are added to the provider WSD. The operations appear in the Operations tab and are also added to each binder contained in the provider WSD.

   If a service signature does not meet the style/use signature requirements established by the existing binder, Designer does not add the service as an operation.

Designer adds the new operation to all binders in the web service descriptor.

4. Click **File > Save**.

   If the operation already exists in the web service descriptor, Designer adds it as a copy and appends "_*n*" to its name, where n is an incremental number.

Tip:
You can also add operations by selecting one or more services in Package Navigator view and dragging them into the Operations tab.

Adding an Operation from another Provider Web Service Descriptor

You can copy or move an operation from another web service descriptor to a service first provider WSD.

Keep the following points in mind when copying or moving an operation from one provider WSD to another:

- You can add operations to a service first provider WSD only.
- Integration Server and Designer do not support mixed "use" across binders and operations and mixed "style" across binders in a single web service descriptor. If the service signature associated with the operation does not meet the style/use signature requirements established by the existing binder, Designer will not add the operation.

To copy or move an existing operation from one provider web service descriptor to another

1. In Package Navigator view, open and lock the provider WSD that contains the operation you want to copy or move.

2. In the Operations tab, select one or more operations. Click or on the web service descriptor editor toolbar.

3. In Package Navigator view, open and lock the provider WSD into which you want to paste the cut or copied operations (the target provider WSD).

4. In the Operations tab of the target WSD, click on the web service descriptor editor toolbar.

5. Click **File > Save**.

   Designer adds the specified operations to the provider WSD. Designer also adds the operations to all binders in the target web service descriptor exactly as they existed in the source web service descriptor. The binder values for each individual binder apply to the operations within the binders.

   If the operation being added already exists in the provider WSD, Designer adds it as a copy and appends "_*n*" to its name, where "*n*" is an incremental number.

Using a 6.5 SOAP-MSG Style Service as an Operation

In webMethods Integration Server version 6.5, you could expose an IS service as a SOAP-MSG web service. The 6.5 SOAP-MSG style services used the default SOAP processor, specified SOAP version 1.1, and specified a style/use of Document/Literal. You can migrate 6.5 SOAP-MSG style services to the web service descriptor framework introduced in Integration Server 7.1 by adding the service as an operation to a provider WSD. By migrating the service, you can leverage the inherent functionality of a provider WSD, such as headers, handlers, faults, and WS-Security.

By default, Integration Server derives the input and output signatures for operations from the services used to create the operation. Integration Server 6.5 required that an IS service in the SOAP-MSG style use a signature that took a *soapRequestData* object and a *soapResponseData* object as input and produced a *soapResponseData* object as output. This signature requirement does not result in meaningful signature information for the operation in WSDL documents generated for the provider WSD. To produce a meaningful, descriptive signature for an operation that corresponds to a 6.5 SOAP-MSG style service, you must select an IS document type or an XML schema element declaration to represent the service input and output signature.

Keep the following points in mind when adding a 6.5 SOAP-MSG style service as an operation to a provider WSD:

- You can add the 6.5 IS service to an existing provider WSD or create a new provider WSD for the IS service. For information about adding an IS service as an operation to a provider WSD, see "Adding an IS Service as an Operation" on page 800. For information about creating a service first provider WSD, see "Creating a Service First Provider Web Service Descriptor" on page 748.
- The provider WSD must have a single binder with the following properties:
  - SOAP version = SOAP 1.1 protocol
  - SOAP binding style = document
  - SOAP binding use = literal
- To produce a meaningful signature for the operation in a WSDL document, you must select an IS document type or an XML schema element declaration to represent the input and output signatures. For information about changing the input or output signature for an operation, a provider WSD, see "Modifying the Signature of a 6.5 SOAP-MSG Style Operation" on page 803.
- If you use an IS document type for the input and/or output signature, the IS document type must satisfy the service signature requirements for the SOAP-MSG style as specified in the *Web Services Developer's Guide* version 6.5.
- If you add any headers to the operation, any existing clients for the 6.5 service must be modified to include the header in the SOAP request.
- Any header handler processing that changes the SOAP message and occurs before service invocation affects the SOAP message passed to the service. Note that 6.5 SOAP-MSG style services expect the SOAP message to be in a certain format. Specifically, any changes to the SOAP body might affect the ability of the 6.5 SOAP-MSG style service to process the request.

- When a 6.5 SOAP-MSG style service is added as an operation, you can add fault processing to the operation response. For fault processing to work, you need to modify the 6.5 SOAP-MSG style service to detect a Fault condition, add Fault output data to the pipeline, and drop the SOAP response message (*soapResponseData* object) from the pipeline.

Modifying the Signature of a 6.5 SOAP-MSG Style Operation

A web service *requires* the input parameters from a signature and *produces* the output parameters. By default, an operation derives the input and output signatures from the services used to create the operation. However, in the case of a 6.5 SOAP-MSG style service, the input and output signatures consist of a *soapRequestData* and *soapResponseData* objects. In a WSDL document, this would result in an vague, meaningless signature. To create a meaningful service signature for a 6.5 SOAP-MSG style operation, you can override the original service signature by selecting an element declaration in an XML schema definition or an IS document type as the service signature. Overriding the service signature is necessary after adding a 6.5 SOAP-MSG style service as an operation to a provider WSD.

Keep the following points when modifying the operation signature source:

- You can only modify the operation signature source in a provider WSD that was created from an IS service. You cannot add or modify the signature of a provider WSD created from a WSDL URL or a UDDI Registry.
- The XML schema definition you select must be located on the web and must be network accessible to consumers of the WSDL. Do not use a local file URL to refer to an external schema.
- If you use an IS document type as the signature for an operation that corresponds to an Integration Server 6.5 SOAP message service, the IS document type must satisfy the service signature requirements for the SOAP MSG protocol a specified in the *Web Services Developer's Guide* version 6.5. For more information about adding an IS 6.5 SOAP message service as an operation, see "Using a 6.5 SOAP-MSG Style Service as an Operation" on page 802.
- An IS document type used to represent the input or output signature of an operation cannot contain top-level fields named "*body" or top-level fields starting with "@".

To modify the signature type of a 6.5 SOAP-MSG style operation

1. In Package Navigator view, open and lock the provider web service descriptor containing the operation whose signature you want to modify.
2. In the Operations tab, select and expand the operation whose signature you want to modify.
3. Do one of the following:

To change the...	Do this...
Input signature	Expand Request and select the Body element.
Output signature	Expand Response and select the Body element.

4. In the Properties view, next to the **Signature** field, click the browse button.

5. In the Modify I/O Signature dialog box, do one of the following:

Select...	To...
Original IS service	Use the input or output signature from the originating IS service as the input or output signature. This is the default.
Existing external XML schema	Use an element declaration from an XML schema definition as the input or output signature.
Document type	Use an IS document type as the input or output signature.

6. If you selected **Existing external XML schema**, do the following:

   a. In the **URL** field, after `http://`, type the web location and name of the XML schema definition that contains the element declaration you want to use to describe the signature.

   b. Click **Load**. Designer displays the global element declarations in the XML Schema.

   c. Select the global element declaration for the input or output signature.

7. If you selected **Document type**, select the IS document type that you want to use to represent the input or output signature.

8. Click **OK**.

9. Click **File > Save**.

   If you selected **Existing external XML schema**, Integration Server automatically uses the internal schema parser to validate the schema. If the **Validate schema using Xerces** property is set to True for the web service descriptor, Integration Server also validates the specified XML Schema definition using the Xerces Java parser. If either parser determines that the schema is invalid, Designer does not save the web service descriptor and displays the validation errors.

Deleting Operations

Keep the following points in mind when deleting operations from a web service descriptor:

- You can delete operations from a service first provider WSD only.
- When you delete an operation on the Operations tab, Designer removes the operation from all the binders in the provider WSD.
- If you delete an operation from within a binder (that is, you delete the operation in the Binders tab), any other instances of that operation in other binders remain in the web service descriptor.

If an operation exists in only one binder and is deleted from that binder, the operation is removed from the web service descriptor.

To delete an operation from a provider web service descriptor

1. In Package Navigator view, open and lock the provider WSD that contains the operation to delete.
2. In the Operations tab, select the operation to be deleted.
3. Click on the web service descriptor editor toolbar. Designer deletes the selected operation from the web service descriptor.
4. Click **File > Save**.

Viewing the Operation Input and Output

You can view the input or output signature of an operation side-by-side with the operations in a web service descriptor.

To view the operation input or output

1. In Package Navigator view, open the web service descriptor.
2. In the Operations tab, navigate to and select the Body element in the Request or Response for an operation.

   For a request, Designer displays the operation input. For a response, Designer displays the operation output.

Adding Headers to an Operation

You can add headers to an operation to incorporate additional processing or functionality for the SOAP message. A *header element* defines the format of the SOAP headers that may be present in a SOAP message (request or response). Headers are optional and can be added to or deleted the request or response in an operation.

For a service first provider web service descriptor, you can:

- Add new headers
- Edit any headers
- Delete any headers

For a WSDL first provider web service descriptor, you can

- Add new headers

- Edit any headers that you add
- Delete any headers that you add or any headers derived from the source WSDL
- Edit the **Must Understand** and **Role** properties for headers derived from the source WSDL

For a consumer web service descriptor, you can:

- Add new headers to the request or response
- Edit any headers that you add
- Delete any headers that you add
- Edit the **Must Understand** and **Role** properties for headers derived from the source WSDL

Note: Integration Server considers all of the headers defined in a web service descriptor to be required. If the header does not exist in the SOAP message at run time, Integration Server throws an error.

While failure when a required header is missing is the correct behavior, Integration Server provides a configuration property to control whether missing required headers in a SOAP response results in an error. If you do not want Integration Server to throw an error in case of missing required headers, set the watt.server.SOAP.ignoreMissingResponseHeader server configuration parameter to true.

Adding a Header to an Operation

Keep the following points in mind when adding headers to operations:

- You can copy or move header document types between headers or between faults, but not between a header and a fault. You can use the same document type for a request and a response, subject to the handlers available in the web service descriptor.
- When adding a header element to a provider web service descriptor, be sure that the header does *not* have the same name as any of the fault elements for that web service descriptor.
- An IS document type used as a header or fault for an operation with a binding style/use of RPC/Encoded cannot contain fields named *body or @attribute fields (fields starting with the "@" symbol).
- You must set up a package dependency if you use an IS document type from a different package as a header.
- A header *must* have a registered header handler. However, you can add the header to an operation and register a header handler for it later. A header without a handler will be ignored or will cause the request to fail (depending on whether the **Must Understand** property for the header is set to **False** or **True**).
- After a header handler is registered in Integration Server, the IS document types associated with the handler will be listed in the selection dialog box that is displayed when you add a header. For more information about registering handlers, see the *Web Services Developer's Guide*.

- The WS Security Handler does not expose supported headers.
- If you add a response header to an operation that uses an In-Only Message Exchange Pattern (MEP), the MEP will change to In-Out MEP. For more information about message exchange patterns, see the *Web Services Developer's Guide*.
- You can also add headers to an operation by dragging IS document types from the Package Navigator view to the Operations tab.
- Integration Server considers all of the headers defined in a web service descriptor to be required.

To add a header to an operation

1. In Package Navigator view, open and lock the web service descriptor to which you want to add a header.
2. In the Operations tab, expand the operation and the request or response to which you want to add the header.
3. Select the header icon and click (**Add Header or Fault**) on the web service descriptor editor toolbar.

   Because a header was selected when you clicked this button, the document selection dialog box displays *only* those IS document types supported by the header handlers listed in the Handlers tab.
4. Select the IS document type to use as a header. Click **OK**.
5. Click **File > Save**.

Important:
When you add a header (or a fault) to a consumer web service descriptor, you must refresh the web service connector(s). See "Refreshing a Web Service Connector" on page 769.

About SOAP Fault Processing

If an error occurs while processing a SOAP request, the response returned to the web service client contains a SOAP fault. You can have the endpoint service signal a fault using one of the following methods:

- Specify a fault whose structure is defined by a fault element, using the fault reasons, code, subcodes, node and role that Integration Server generates.

  At design time, you can identify the structure of SOAP faults with which an operation can respond by adding fault elements to the operation response in a web service descriptor. Fault elements are optional and can be added to any web service descriptor. For more information, see "About SOAP Fault Elements" on page 809.

To signal a fault that uses one of the fault elements, set up the endpoint service for the operation so that it places an instance document of one of the fault elements into the top level of the service pipeline. The name of the instance document *must* match the name assigned to the fault element. Integration Server recognizes the fault document in the pipeline, and when generating the fault detail, uses the IS document type defined in the fault element for the structure of the instance document. If the document has a name that matches a fault element, but a different structure, unexpected results will occur.

Integration Server generates a SOAP response that contains a SOAP fault. The SOAP fault contains the detail from the instance document and uses fault reasons, code, subcodes, node and role that Integration Server generates.

- Specify a fault whose structure is defined by a fault element, but override the fault reasons, code, subcodes, node and/or role that Integration Server generates.

  Set up the endpoint service for an operation so that it places an instance document of one of the defined fault elements into the *$fault/detail* variable. The name of the instance document *must* match the name assigned to the fault element. Integration Server recognizes the fault document in the *$fault/detail* variable, and when generating the fault detail, uses the IS document type defined in the fault element for the structure of the instance document. If the document has a name that matches a fault element, but a different structure, unexpected results will occur.

  To override the fault reasons, code, subcodes, node and/or role, set up the endpoint service to also provide the corresponding values in fields within the *$fault* variable. For a description of the *$fault* variable, see "The $fault Variable" on page 811.

  Integration Server recognizes the *$fault* variable in the pipeline. Subsequently, Integration Server generates a SOAP response that contains a SOAP fault using the information from the *$fault/detail* variable. The SOAP fault contains the detail from the instance document and uses values specified for fault reasons, code, subcodes, node and/or role within the *$fault* variable to override the corresponding values that Integration Server generates.

  Note:
  If there is a top-level instance document for the fault, in addition to the one in the *$fault/detail* variable, Integration Server ignores the top-level document.

- Specify a fault with a structure that was not previously defined using a fault element. Optionally, override the fault reasons, code, subcodes, node and/or role that Integration Server generates.

  Although you can identify the structure of SOAP faults in advance, it is not required. To signal a fault at run time, you can add fault information that does not match defined fault elements to the *$fault/detail* variable in the pipeline. Be sure that the name does *not* match any defined fault elements. Integration Server recognizes the *$fault/detail* variable in the service pipeline. Because the document in the *$fault/detail* variable does not match a defined fault element, Integration Server generates the fault detail without using an IS document type for the structure.

  To override the fault reasons, code, subcodes, node and/or role, set up the endpoint service to provide the corresponding values in fields within the *$fault* variable. For more information, see "The $fault Variable" on page 811.

Integration Server ignores any top-level instance document that might be in the pipeline for a fault. Using the information from the *$fault/detail* variable, Integration Server generates a SOAP response that contains a SOAP fault. If values are specified for the fault reasons, code, subcodes, node and/or role within the *$fault* variable, Integration Server uses those values instead of values it generates.

Additionally, faults can occur for the following reasons:

- The endpoint service throws a service exception.

  In this case, Integration Server constructs a fault message out of the service exception. If the pipeline also contains a *$fault* variable, Integration Server uses the information specified in the *$fault* variable to override the fault information.

  To make the *$fault* variable available, you can write a Java service that throws a ServiceException, but before throwing the exception, places the *$fault* variable in the pipeline.

  Alternatively, for a flow service, you can use the EXIT with failure construct. As a result, before exiting the flow service with a failure, you can place the *$fault* variable into pipeline.

- A request handler service ended in failure and signaled that a fault should be generated.

  When the request handler returns a status code 1 or 2, Integration Server generates a SOAP fault, along with the fault code, subcodes, reasons, node, and role for the fault. You can use the pub.soap.handler:updateFaultBlock service to modify the code, subcodes, reasons, node, and/or role that Integration Server generates.

  Note:
  When the request handler returns status code 3, you are expected to build the SOAP fault. As a result, the pub.soap.handler:updateFaultBlock service is not necessary.

  You can invoke the pub.soap.handler:updateFaultBlock service in a response or fault handler to update the fault created due to the failure in the request handler chain. For more information about using the service, see "Modifying a Returned SOAP Fault" on page 813. For more information about handlers, see the *Web Services Developer's Guide*.

About SOAP Fault Elements

To identify the information to provide in a SOAP fault at design time, you add fault elements to the operation response in a web service descriptor. The fault element, which is an IS document type, describes the expected structure of the Detail element in the SOAP fault. Fault elements are optional and can be added to any web service descriptor.

When you create a service first provider web service descriptor, add fault elements to represent the SOAP faults that an operation in the web service descriptor might return. If an error occurs at run time, the underlying service that corresponds to the operation can signal a fault by returning an instance document for one of the IS document types used as a fault element. Integration Server recognizes the fault document in the service pipeline and subsequently generates a SOAP response that contains a SOAP fault. Within the SOAP fault, the Detail element contains the fault document.

When you create a WSDL first provider web service descriptor or a consumer web service descriptor, Integration Server creates an IS document type for each message element in the source

WSDL document. If an operation in a WSDL defines a soap:fault element, Integration Server generates an IS document type for the fault element.

In a consumer web service descriptor, the web service connector that corresponds to the operation includes logic to detect the fault element in the SOAP response. Integration Server then places the contents of the fault document into the detail document in the output parameter. The structure of the detail element matches the structure of the IS document type used as the fault element.

Note:
The structure of the SOAP fault returned by the web service connector depends on the version of Integration Server on which the web service descriptor was created. For more information, see *Web Services Developer's Guide*.

It is possible for a web service to return a fault that does not appear in a WSDL file. To account for these SOAP faults, you can add fault elements to a WSDL first provider web service descriptor or a consumer web service descriptor. For more information, see "Adding a Fault Element to an Operation" on page 810.

Adding a Fault Element to an Operation

Keep the following points in mind when adding fault elements to an operation:

- You add fault elements to an operation response.
- The fault document must be an IS document type.
- You must set up a package dependency if you use an IS document type from a different package as a fault.
- If you add a fault to an operation that uses an In-Only Message Exchange Pattern (MEP), the MEP will change to Robust In-Only MEP. For more information about message exchange patterns, see the *Web Services Developer's Guide*.

To add a fault element to an operation

1. In Package Navigator view, open and lock the web service descriptor to which you want to add a fault element.

2. In the Operations tab, expand the operation and the response to which you want to add the fault element.

3. Select the **Fault** icon and click (**Add Header or Fault** button) on the web service descriptor editor toolbar.

   Because a fault was selected when you clicked this button, Designer displays the default document selector dialog.

4. Select the IS document type to use as the fault element. Click **OK**.

5. If you want to change the name of the fault element, with the fault element selected, in the **General** category of the Properties view, update the **Name** property.

6. Click **File > Save**.

Important:
When you add a fault to a consumer web service descriptor, you must refresh the web service connector(s). See "Refreshing a Web Service Connector" on page 769.

Notes:

- If you add a fault element to an operation in a consumer web service descriptor, and then refresh the web service connector, Integration Server updates the logic of the web service connector to look for and handle the fault at run time.
- If you add a fault element to an operation in a WSDL first provider web service descriptor, the WSDL document generated from the provider web service descriptor will include the new faults as soap:fault elements in the operation.
- You can add multiple fault elements to an operation in a web service descriptor. At run time, if the service that corresponds to the operation returns multiple fault documents, the SOAP fault in the resulting SOAP response will contain only one fault document. Specifically, Integration Server returns the fault document that is an instance of the IS document type that appears first in the operations list of fault elements.

  For example, suppose that an operation had three fault elements listed in this order: faultA, faultB, and faultC. Note that each fault element corresponds to an IS document type of the same name. At run time, execution of operation (service) results in two fault documents—one for faultB and one for faultC. In the SOAP response generated by Integration Server, the SOAP fault contains the faultB document only.

The $fault Variable

Use the *$fault* variable to override values Integration Server generates for a fault. To do so, specify the fault detail in the *$fault/detail* variable. Then, to override the fault reasons, code, subcodes, node and/or role, provide the corresponding values.

The following shows the structure of the *$fault* variable.

Variable	Description
$fault	**Document** Fault information that overrides other fault information in the service pipeline, if any.
code	**Document** Optional. The fault code and possible subcodes. Integration Server uses values you specify to modify the fault code and subcodes it generates for the fault.

Merk:

Variable	Description
	For a SOAP 1.1 fault, Integration Server ignores any values specified for *subcodes*.
	namespaceName **String** The namespace name for the SOAP fault code.
	localName **String** A code that identifies the fault.
	subcodes **Document List** Optional. Subcodes that provide further detail. Each Document in the *subCodes* Document List contains: ■ *namespaceName* for the subcode ■ *localName* that identifies the subcode
reasons	**Document List** Optional. Reasons for the SOAP fault. Integration Server uses values you specify to modify the reasons it generates for the fault.
	Merk: For a SOAP 1.1 fault, if you specify more than one reason, Integration Server uses the first reason. Multiple reasons are supported for SOAP 1.2 faults.
	@lang **String** Language for the human readable description.
	body* **String Text explaining the cause of the fault.
node	**String** Optional. The URI to the SOAP node where the fault occurred. Integration Server uses value you specify to modify the node it generates for the fault.
	Merk: For a SOAP 1.1 fault, Integration Server ignores any values specified for *node*.
role	**String** Optional. The role in which the node was operating at the point the fault occurred. Integration Server uses value you specify to modify the role it generates for the fault.
detail	**Document** Fault information you want Integration Server to use. This overrides any top-level instance document that defines the fault detail.

Modifying a Returned SOAP Fault

If a provider or consumer web service results in a fault, you can modify values in the fault, if needed. For example, you might want to alter the fault code if your error handling requires a specific code.

To update a SOAP fault, use the pub.soap.handler:updateFaultBlock service. For more information about this service, see the *webMethods Integration Server Built-In Services Reference*. You can invoke the updateFaultBlock service from a response handler or fault handler service for a web service provider. Use the service to customize one or more of the following SOAP fault fields:

Fault field you can customize	Notes
fault code and subcodes	For a SOAP 1.1 fault, if you specify subcode values, the service ignores them because subcodes are only applicable for a SOAP 1.2 fault.
fault reasons	For a SOAP 1.1 fault, if you specify more than one reason, the service only uses the first reason. Multiple reasons are supported for SOAP 1.2 faults.
fault node	For a SOAP 1.1 fault, if you specify a value for node, the service ignores it because the fault node is only applicable for a SOAP 1.2 fault.
fault role	The fault role is supported for both SOAP 1.1 and SOAP 1.2 faults.

Viewing Document Types for a Header or Fault Element

You can view an IS document type used for a header or fault element side-by-side with the operations for a web service descriptor. The IS document type is read-only in the Operations tab.

To view the document type for a header or fault element in an operation

1. In Package Navigator view, open the web service descriptor.

2. In the Operations tab, navigate to and select the header or fault element for which you want to view the IS document type contents.

Working with Handlers

When working with web services on Integration Server, the SOAP body portion of the SOAP message contains the data representing the input and output signatures of the underlying SOAP operation. In typical processing, Integration Server converts the SOAP body between its XML representation in the SOAP message and the Document (IData) representation used within Integration Server automatically.

In addition to the data contained in the SOAP body, a SOAP message might contain data in the SOAP headers. The best way to access the SOAP headers is to use handlers. A handler, sometimes called a header handler, provides access to the entire SOAP message.

Handlers can be used to perform various types of processing, including processing SOAP headers, adding SOAP headers, removing SOAP headers, passing data from the header to the endpoint service or vice versa.

In Integration Server, a handler is a set of up to three handler services. The handler can contain one of each of the following handler services:

- Request handler service
- Response handler service
- Fault handler service

For detailed information about request, response, or fault handler services, see *Web Services Developer's Guide*.

Any IS service can be used as a handler service. However, handler services must use a specific service signature. Integration Server defines the service handler signature in the pub.soap.handler:handlerSpec specification. Integration Server also provides several services that you can use when creating handler services. These services are located in the pub.soap.handler folder in the WmPublic package.

When you register a handler, you name the handler, identify the services that function as the request, response or fault handler services, and indicate whether the handler is for use with provider web service descriptors or consumer web service descriptors.

You can assign multiple handlers to a web service descriptor. Designer displays the handlers on the Handlers tab. The collection of handlers assigned to a web service descriptor is called a handler chain. For a consumer web service descriptor, Integration Server executes the handler chain for output SOAP requests and inbound SOAP responses. For a provider web service descriptor, Integration Server executes the handler chain for inbound SOAP requests and outbound SOAP responses.

When executing the handler chain, Integration Server executes request handler services by working through the handler chain from top to bottom. However, Integration Server executes response handler services and fault handler services from bottom to top.

The order of handlers in the handler chain may be important, depending on what processing the handlers are performing.

Setting Up a Header Handler

To create and implement a header handler, you need to:

- Build the services for handling a request, handling a response, and handling a fault. Use the pub.soap.handler:handlerSpec specification as the signature for a service that acts as a header handler.
- Register the combination of those services as a header handler.

- Assign the header handler to the web service descriptor.

Registering a Header Handler

Register the handler as either a consumer or provider using pub.soap.handler:registerWmConsumer or pub.soap.handler:registerWmProvider, respectively. During registration:

- You provide a logical name for the handler.
- You specify the services for handling a request, a response, and a fault as input.
- You optionally specify the list of QNames on which the handler operates.

  Specify QNames only if you want to associate with handler with one or more QNames. Registering QNames with a handler provides the following benefits:

  - Integration Server can perform mustUnderstand checking for the header with the QName at run time. If a service receives a SOAP message in which a header requires mustUnderstand processing by the recipient, Integration Server uses the header QName to locate the handler that processes the header. Note that the handler must be part of the handler chain for the WSD that contains the service.
  - When adding headers to a WSD, Designer populates the list of IS document types that can be used as headers in the WSD with the IS document types whose QNames were registered with the handlers already added to the WSD. If you add a IS document type as a header to a WSD and the QName of that IS document type is not associated with a handler, Designer adds the header but display a warning stating that there is not an associated handler.
  - When consuming WSDL to create a provider or consumer WSD, Integration Server automatically adds a handler to the resulting WSD if the WSDL contains a QName supported by the handler.

Note: Integration Server stores information about registered header handlers in memory. Integration Server does not persist registered header handler information across restarts. Consequently, you must register header handlers each time Integration Server starts. To accomplish this, create a service that registers a header handler and make that service a start up service for the package that contains the services that act as header handlers.

Adding a Handler to a Web Service Descriptor

To add a handler to a web service descriptor

1. In Package Navigator view, open and lock the web service descriptor to which you want to add handlers.

2. In the Handlers tab, click on the web service descriptor toolbar. Or right-click and select **Add Handler**.

3. Select the registered handler that you want to add to the web service descriptor.

4. Click **File > Save**.

5. Once a handler is added to a web service descriptor, you may optionally add any headers associated with the handler to the request or response elements of operations within the web service descriptor.

   Note:
   You must set up a package dependency if the web service descriptor uses a handler from a different package.

Deleting a Handler from a Web Service Descriptor

To delete a handler from a web service descriptor

1. In Package Navigator view, open and lock the web service descriptor from which you want to remove handlers.

2. Click the Handlers tab.

3. Select the handler that you want to delete.

4. Click on the web service descriptor toolbar or right-click and select **Delete**.

   Designer removes the selected handler is deleted from the Handlers tab and from the web service descriptor. If the web service descriptor still contains a header associated with the deleted handler, Designer displays a warning.

Working with Policies

WS-Policy is a model and syntax you can use to communicate the policies associated with a web service. *Policies* describe the requirements, preferences, or capabilities of a web service. You attach a policy to a *policy subject*, for example, a service, endpoint, operation, or message. After attaching a policy to a policy subject, the policy subject becomes governed by that policy.

webMethods Integration Server provides support for *Web Services Policy Framework* (WS-Policy) Version 1.2.

In Integration Server, a policy is specified using policy expressions in an XML file called a *WS-Policy file* (more simply called *policy file*). Integration Server comes with some policy files out of the box. Additionally, you can also create policy files. For more information, see *Web Services Developer's Guide*.

To have a web service governed by the policy in a WS-Policy file, you attach the policy to the web service descriptor. You can attach WS-Policies at the binding operation message type level, such as input, output, and fault, in a web service descriptor.

Attaching a Policy to a Web Service Descriptor

You can attach one or more policies to a single web service descriptor. Also, multiple web service descriptors can share the same policy. You can attach any policy that resides in the policy repository.

Keep the following points in mind when attaching policies to a web service descriptor:

- To attach a policy to a web service descriptor, the **Pre-8.2 compatibility mode** property of the web service descriptor must be set to false.

  If you change the **Pre-8.2 compatibility mode** property of a web service descriptor from false to true after a policy is attached to it, the policy subject will no longer be governed by that policy.

  For more information about **Pre-8.2 compatibility mode** property, see "About Pre-8.2 Compatibility Mode" on page 818.

- Policies are attached to a binder, specifically the input, output, and fault messages for the operations in a binder. A policy attached to a binder applies to all of the operations in the binder. Integration Server and Designer do not support applying policies to some operations in a binder and not others. For example, suppose a binder has two operations and you attach the Addressing policy to the binder, selecting Input and Output as the messages type to which the policy applies. The input and output messages for both operations in the binder are governed by the Addressing policy.

- When attaching policies, avoid attaching a policy that contains policy assertions that Integration Server does not support. For information about supported assertions, see the *Web Services Developer's Guide*. If you attach a policy that contains unsupported policy assertions, unexpected behavior may occur.

- If you attach a policy to a WSDL first provider web service descriptor or a consumer web service descriptor, the attached policy will override any annotated policy in the source WSDL.

  - For a web service descriptor with a policy attached to it, the attached policy always takes precedence at run time.

  - For a consumer web service descriptor, even though the consumer WSDL will not show the attached policy, Integration Server will enforce the attached policy at run time.

- When you attach a policy to or remove a policy from a provider web service descriptor, the WSDL generated for that web service descriptor is changed as well. Any web service clients generated from the WSDL will need to be regenerated.

- When you attach a policy to or remove a policy from a consumer web service descriptor, you do not need to refresh the web service connectors to pick up the policy change. Integration Server detects and enforces the policy change at run time.

- If the policy you are attaching contains WS-SecurityPolicy assertions and you also want to use MTOM streaming, be aware that if the fields to be streamed are also being signed and/or encrypted, Integration Server cannot use MTOM streaming because Integration Server needs to keep the entire message in memory to sign and/or encrypt the message.

To attach a policy to a web service descriptor

1. In Package Navigator view, open and lock the web service descriptor to which you want to attach a policy.

2. In the Policies tab, click on the web service descriptor toolbar, or right-click and select **Attach Policy**.

3. Select the policies that you want to attach to the web service descriptor.

   Designer displays the policies that you selected in the Policies tab.

4. Against each policy in the Policies tab, select the appropriate check boxes to attach a WS-Policy to **Input**, **Output**, and/or **Fault** message type. You can choose to attach a policy to all message types or to any of the three message types. You must select at least one message type for each policy in the Policies tab if you want the web service descriptor to be governed by that policy.

   Note:
   By default, all the three message types are selected.

5. Click **File > Save**.

Removing a Policy from a Web Service Descriptor

If you no longer want a web service descriptor to be governed by a particular policy, you can detach or remove the policy from the web service descriptor.

To remove a policy from a web service descriptor

1. In Package Navigator view, open and lock the web service descriptor from which you want to remove a policy.

2. Click the Policies tab.

3. Select the policy that you want to delete from the web service descriptor.

4. Click on the web service descriptor toolbar or right-click and select **Delete**.

About Pre-8.2 Compatibility Mode

Integration Server version 8.2 introduces support for web service features, such as SOAP over JMS, and WS-Policy based WS-Security configuration, that are available through the current web services stack. Some of the features and behavior included in the current web services stack are not compatible with the features and run-time behavior of web service descriptors created on the

earlier version of the web services stack, specifically the web services stack available in Integration Server versions 7.x, 8.0, and 8.0 SP1.

To ensure that web service descriptors developed on the earlier version of the web services stack execute as expected, web service descriptors now have a **Pre-8.2 compatibility mode** property. This property determines the version of the web services stack on which the web service descriptor runs. The version of the web service stack used by the web service descriptor determines the design-time features and run-time behavior for the web service descriptor. The value of the **Pre-8.2 compatibility mode** property indicates the version of the web service stack with which the web service descriptor is compatible:

- When the **Pre-8.2 compatibility mode** property is set to true, the web service descriptor runs on the earlier version of the web services stack, specifically the web services stack available in Integration Server versions 7.x, 8.0, and 8.0 SP1. Web service descriptors running in pre-8.2 compatibility mode have the same design-time features and run-time behavior as web service descriptors run in versions of Integration Server prior to version 8.2.
- When the **Pre-8.2 compatibility mode** property is set to false, the web service descriptor runs on the current version of the web services stack. web service descriptors that do not run in pre-8.2 compatibility mode have the design-time features and run-time behavior available in the current version of the web services stack.

Note:
You can use Designer 8.2 or later with an Integration Server 8.2 or later to create and edit a web service descriptor regardless of the compatibility mode.

Setting Compatibility Mode

Keep the following points in mind when setting the **Pre-8.2 compatibility mode** property for a web service descriptor:

- You can set the compatibility mode using Designer 8.2 or later only.
- The compatibility mode alters the design-time features available for the web service descriptor and might change the run-time behavior of the web service descriptor.
- You can use the pub.utils.ws:setCompatibilityModeFalse service to change the **Pre-8.2 compatibility mode** property value for multiple web service descriptors at one time. For more information, see the *webMethods Integration Server Built-In Services Reference*.
- If you intend to change the compatibility mode of a web service descriptor for which you published metadata to CentraSite, first retract metadata for the web service descriptor. Next, change the compatibility mode. Finally, republish metadata for the web service descriptor to CentraSite.

To set the compatibility mode for a web service descriptor

1. Open Designer.

2. In Package Navigator view, open and lock the web service descriptor for which you want to change the compatibility mode.

3. In the Properties view, next to **Pre-8.2 compatibility mode**, do one of the following:

   - Select **True** if you want the web service descriptor to run in pre-8.2 compatibility mode. **True** indicates that Integration Server will deploy the web service descriptor to the web services stack available in Integration Server versions 7.x, 8.0, and 8.0 SP1.
   - Select **False** if you do not want the web service descriptor to run in pre-8.2 compatibility mode. **False** indicates that Integration Server will deploy the web service descriptor to the web services stack available in Integration Server 8.2 or later.

   Designer verifies that the web service descriptor can be deployed to the web services stack that corresponds to the chosen compatibility mode and displays any errors or warnings.

4. If Designer displays errors or warnings, do one of the following:

   - If errors occur, Designer determined that the web service descriptor cannot be deployed to the corresponding web services stack successfully. Designer displays the errors that identify the functionality that is incompatible with the web services stack. Click **OK** to cancel the change to the Pre-8.2 compatibility mode property.
   - If warnings occur, Designer determined that the web service descriptor can be deployed to the corresponding web services stack successfully but some run-time behavior might change. Designer displays any warnings about the functional changes of the web service descriptor in the web services stack. Click **OK** to proceed with the change to the **Pre-8.2 compatibility mode** property. Click **Cancel** to cancel the change.

5. Click **File > Save**.

Features Impacted by Compatibility Mode

The following table identifies the web service features impacted by the compatibility mode of the web service descriptor.

Name	Description	Behavior
Binders	Multiple binders with different operations	In pre-8.2 compatibility mode, Integration Server permits a single web service descriptor to contain multiple binders with different operations. When not in pre-8.2 compatibility mode, all the binders for a single web service descriptor must contain the same operations. Integration Server will not save a service first provider web service descriptor if it contains multiple binders that list different operations.

Name	Description	Behavior
Binding Styles	Mixed binding styles in web service descriptors	In pre-8.2 compatibility mode, Integration Server does not restrict the binding styles in a web service descriptor. A web service descriptor could contain binders that used different binding styles. When not in pre-8.2 compatibility mode, Integration Server requires all binders within a web service descriptor to use the same binding style. That is, all the binders in a single web service descriptor should specify a binding style of RPC or Document for all of the binders.
Binding Styles	Mixed binding styles in WSDLs	In pre-8.2 compatibility mode, Integration Server creates web service descriptors from WSDLs that contained bindings that used different binding styles. When not in pre-8.2 compatibility mode, Integration Server requires all bindings in the consumed WSDL to have the same style. Integration Server will not create a web service descriptor from a WSDL that contains mixed styles across its bindings.
JMS Bindings	Support for JMS bindings and JMS binders	In pre-8.2 compatibility mode, Integration Server supports HTTP and HTTPS bindings only. ■ Integration Server will not create a WSDL first provider web service descriptor from a WSDL document that contains a JMS binding. ■ When creating a consumer web service descriptor, Integration Server will not create binders that correspond to JMS bindings in the WSDL document. If the WSDL document contains only JMS bindings, Integration Server will not create the consumer web service descriptor. ■ A web service descriptor cannot contain any binders that specify the JMS transport. When not in pre-8.2 compatibility mode, Integration Server supports JMS bindings in addition to HTTP and HTTPS bindings. ■ Integration Server can create provider web service descriptors from WSDL documents that contain JMS bindings. ■ When creating a consumer web service descriptor, Integration Server will create binders that

Name	Description	Behavior
		correspond to JMS bindings in the WSDL document. ■ A web service descriptor can use binders that specify JMS as the transport.
Message Exchange Pattern (MEP) Support	Added In-Only MEP and Robust In-Only MEP support	In pre-8.2 compatibility mode, Integration Server supports only In-Out MEP for web service operations. When not in pre-8.2 compatibility mode, Integration Server supports In-Only MEP and Robust In-Only MEP, in addition to In-Out MEP.
MTOM Attachments	Streaming MTOM Attachments	In Pre-8.2 compatibility mode, Integration Server supports MTOM attachments in both inbound and outbound messages. However, Integration Server cannot stream the MTOM attachments. Integration Server always holds the MTOM attachment in memory. When not in Pre-8.2 compatibility mode, Integration Server can stream the MTOM attachments for both inbound and outbound SOAP messages.
Port types	Port types in web service descriptors	In pre-8.2 compatibility mode, Integration Server supports multiple port types. When Integration Server generated a WSDL for a multi-binder web service descriptor, the resulting WSDL had multiple port types. When not in pre-8.2 compatibility mode, Integration Server does not support multiple port types. As a result, when Integration Server generates a WSDL for a multi-binder provider web service descriptor, it changes the port type names so that the generated WSDL has only a single port type.
Port types	Port types in WSDLs	In pre-8.2 compatibility mode, Integration Server creates web service descriptors from WSDLs that contained multiple port types. When not in pre-8.2 compatibility mode, Integration Server does not support creating web service descriptors from WSDLs that contain multiple port types.
Provider web service descriptor	Provider web service descriptor with no operations	In pre-8.2 compatibility mode, Integration Server permits a provider web service descriptor that does not contain any operations.

Name	Description	Behavior
		When not in pre-8.2 compatibility mode, Designer will not save a web service descriptor if it does not contain at least one operation.
Response services	Response services in consumer web service descriptor	In pre-8.2 compatibility mode, Integration Server does not create the responseServices folder when creating a consumer web service descriptor. When not in pre-8.2 compatibility mode, Integration Server creates the responseServices folder that contains response services for each operation in the WSDL document and a genericFault_Response service when creating a consumer web service descriptor.
Service name	Name attribute in wsdl:service element	In pre-8.2 compatibility mode, the local name of the web service descriptor determines the value of the name attribute in the wsdl:service element in the associated WSDL document. For example, suppose that a web service descriptor has the fully qualified name folder.myFolder:myWebService. In the WSDL document, the value of the name attribute in the wsdl:service element is "myWebService". When not in pre-8.2 compatibility mode, the fully qualified name of the web service descriptor determines the value of the name attribute in the wsdl:service element. For example, suppose that a web service descriptor has the fully qualified name folder.myFolder:myWebService. In the WSDL document, the value of the name attribute in the wsdl:service element is "folder.myFolder:myWebService".
Web service handlers	Web service handlers based on JAX-RPC	In pre-8.2 compatibility mode, Integration Server supports web service handlers based on JAX-RPC. When not in pre-8.2 compatibility mode, Integration Server will not execute web service handlers based on JAX-RPC.
Web service handlers	Web service handler chain execution	In pre-8.2 compatibility mode, you can use the WS-Security facility to secure a web service. The WS-Security facility secures web services using the WS-Security handler. Because a handler is used, Integration Server can perform additional processing before and after the security processing. That is, for inbound messages Integration Server can invoke handlers before invoking the WS-Security handler to

Name	Description	Behavior
		perform security processing. For outbound messages, Integration Server can invoke custom handlers after it invokes the WS-Security handler, but before it sends the outbound message. The order of the handlers on the Handlers tab determines the order in which Integration Server invokes them. When not in pre-8.2 compatibility mode, for inbound messages, Integration Server always performs the security processing first upon receiving the message. As a result, Integration Server cannot invoke custom handlers before the security processing of an inbound message. For outbound messages, Integration Server always performs the security processing last, right before it sends the message. As a result, Integration Server cannot invoke handlers after the security processing of an outbound message.
WS-Addressing	WS-Addressing	In pre-8.2 compatibility mode, Integration Server does not support WS-Addressing. When not in pre-8.2 compatibility mode, Integration Server implements WS-Addressing by associating WS-Addressing policies that conform to Web Services Policy framework to web service descriptors.
WS-Policy	WS-Policy	In pre-8.2 compatibility mode, Integration Server does not support WS-Policy. When not in pre-8.2 compatibility mode, Integration Server supports the WS-Policy framework.
WS-Security	WS-Security facility	In pre-8.2 compatibility mode, Integration Server implemented WS-Security by associating WS-Security handlers to web service descriptors. When not in pre-8.2 compatibility mode, Integration Server implements WS-Security by associating WS-Security policies that conform to Web Services Policy framework to web service descriptors.

37 Working with UDDI Registry

- Opening UDDI Registry View 826
- Connecting to a UDDI Registry 827
- Disconnecting from a UDDI Registry 828
- Refreshing a UDDI Registry Session 828
- Browsing for Web Services in a UDDI Registry 828
- Creating a Web Service Descriptor from a UDDI Registry 829
- Publishing a Service to UDDI Registry 830
- Deleting a Service from UDDI Registry 831

A UDDI registry (Universal Description, Discovery, and Integration) is an XML-based registry for businesses worldwide to list themselves on the Internet. It allows users to view, find, and share web services.

When working with a UDDI registry from Designer you can:

- Discover the web services published in a UDDI registry.

  Designer displays a list of the web services that are published in a UDDI registry. By default, Designer displays all published services, but you can use a filter to limit the number of services shown.

- Incorporate a web service into Integration Server.

  You can incorporate a web service in the UDDI registry into your integration solution by creating a consumer web service descriptor from the web service. Designer automatically generates a web service connector for each operation in the web service, which can be invoked in the same way as any other IS service. For more information, see *Web Services Developer's Guide*.

- Publish services to a UDDI registry.

  You can make a service that resides on Integration Server (such as a flow service, Java service, C service, or adapter service) available as an operation of a web service and then publish the web service to a UDDI registry.

Opening UDDI Registry View

To connect to a UDDI registry from Designer, you must open the UDDI Registry view.

The UDDI Registry view contains icons to represent the UDDI registry, the registered business entities, and the web services published to the UDDI registry. The UDDI Registry view also contains a toolbar with buttons for performing various operations.

Note:
If your UDDI registry is CentraSite, you will also be able to use the Registry Explorer view in Designer in addition to the UDDI Registry view. The Registry Explorer view displays the contents of the CentraSite registry to which Designer is currently connected. To open the Registry Explorer view, select **Window > Show View > Other** and in the Show View dialog box, select **CentraSite > Registry Explorer**.

To open the UDDI Registry view

1. In Designer: **Window > Show View > Other**.
2. In the Show View dialog box, select **Software AG Service Development > UDDI Registry**.
3. Click **OK**. Designer displays the UDDI Registry view.

Connecting to a UDDI Registry

To use Designer to view or publish web services, you must first connect to a UDDI registry.

Keep the following points in mind when connecting to a UDDI registry:

- Designer supports UDDI v3 only. It will not connect to a registry based on earlier versions of UDDI.
- You must have a valid UDDI v3 registry account, configured with the proper permissions to perform UDDI activities.
- The specified UDDI v3 registry must contain at least one *business entity*. A business entity is a logical grouping of web services, used to represent businesses and providers within UDDI. You must publish each web service within a business entity.
- You can only connect to one UDDI registry at a time from Designer. To connect to another UDDI registry, you must first disconnect from the existing session.
- While connecting to a UDDI registry, if you enter only the Inquiry URL, the UDDI connection will be unauthenticated. You can only view entities or services that are viewable anonymously as per the registry policies. To publish a web service, you must authenticate the connection by providing Security URL, Publish URL, UDDI user ID, and password.

To connect to a UDDI registry

1. In UDDI Registry view, right-click and select **Open UDDI Registry Session**.
2. In the Open UDDI Registry Session dialog box, select **Add New Registry**.
3. In the **Registry Name** field, enter a name for the UDDI registry.
4. Enter information in the following fields as appropriate:

In this field...	Enter
Inquiry URL	The URL configured for browsing the UDDI registry. This field is mandatory.
Security URL	The security URL for the UDDI registry. This field can be mandatory or optional, depending on the registry.
Publish URL	The URL configured for publishing services to the UDDI registry. This field is mandatory if you want to publish a web service descriptor to the UDDI registry.

5. Enter your UDDI registry user ID and password.
6. Click **Finish**.

Disconnecting from a UDDI Registry

- In UDDI Registry view, right-click and select **Close UDDI Registry Session**.

Refreshing a UDDI Registry Session

Designer refreshes the contents of the UDDI Registry view when you create a web service automatically. However, you will need to refresh the UDDI registry session in the following situations:

- Other users add or delete web services from the UDDI registry.
- Designer loses its connection to a UDDI registry.
- The UDDI registry operates with some form of governance and a newly added web service does not appear immediately.

To refresh a UDDI registry session

- In UDDI Registry view, right-click and select **Refresh UDDI Registry Display**. Integration Server refreshes the display of the UDDI registry.

Browsing for Web Services in a UDDI Registry

After you connect to a UDDI registry, you can browse for web services that you would like to incorporate into your integration solution. You can discover web services by scrolling through the UDDI Registry view. By default, the UDDI Registry view displays all services published in the UDDI registry.

To help find a particular web service, you can reduce the number of displayed web services by filtering the contents of the UDDI Registry view based on the value of a specified web service property.

Applying a Filter to UDDI Registry

To make it easier to find a particular web service from the list of all the services published in the UDDI registry, you can filter the contents of the UDDI Registry view.

When applying a filter to the UDDI Registry view, keep the following points in mind:

- Designer continues to apply the specified filter until you explicitly clear the filter. Designer saves the filter string across Designer and UDDI registry sessions.
- If you publish a service that does not meet the criteria specified in the currently applied filter, Designer does not display the newly published web service in the UDDI Registry view.
- Designer applies each filter that you create to the entire contents of the UDDI registry. For example, if you apply two filters in succession, Designer clears the first filter before applying the second filter. Designer does not apply the second filter to the results of the first filter.

To filter the contents of the UDDI Registry view

1. In UDDI Registry view, right-click and select **Filter UDDI Registry Display**.

2. In the Filter UDDI Registry Display dialog box, in the **Enter Filter Text** field, type the text to use as the filter criteria.

   Integration Server treats the text as a partial string. For example, if you enter “vic”, “victory” and “services” will both fit the search criteria. The percent sign (%) can be used for an approximate search within the text. For example, for “a%n”, “ain” and “Amazon” fit the search criteria.

3. Select the check boxes for **WebService Name**, **WebService Description**, or both to specify which properties you want Designer to examine for the **Enter Filter Text** text string.

4. Click **OK** to apply the filter.

Clearing an Applied Filter

If you have applied a filter to limit the contents of the UDDI Registry view, Designer continues to use this filter until you explicitly remove the filter. If the button on the UDDI Registry view toolbar is active, it indicates that Designer is using a filter to limit the displayed web services.

To clear a filter

1. In UDDI Registry view, right-click and select **Clear UDDI Filter**.

2. Click **OK** to confirm removing the filter.

   Designer removes the filter and displays all the published web services in the UDDI registry.

Creating a Web Service Descriptor from a UDDI Registry

You can create a consumer or WSDL first provider web service descriptor from a web service in the UDDI registry.

To create a web service descriptor from a web service in a UDDI registry

1. In UDDI Registry view, right-click and select **Create Web Service Descriptor**.

2. In the New Web Service Descriptor dialog box, select either **Provider (Inbound Request)** or **Consumer (Outbound Request)**.

   Follow the prompts that Designer displays and enter the required information for the type of web service descriptor you are creating. Designer creates the provider web service descriptor

and saves it to the folder you specified. Designer also creates supporting IS elements, such as flow services and IS document types.

Note:
You can also create a web service descriptor by dragging and dropping a service from the UDDI Registry view to a folder in the Package Navigator view.

Publishing a Service to UDDI Registry

For a service (flow, Java, C, adapter) to be available as a web service through a UDDI registry, you must first create a service first provider web service descriptor that uses the IS service as an operation and then publish that web service descriptor to the UDDI registry.

When you use Designer to publish a web service to a UDDI registry, the Integration Server creates a WSDL file and adds it to the registry.

Keep the following points in mind when you publish a service to a UDDI registry using Designer:

- You can publish a service by creating a provider web service descriptor for the service and dragging it directly from the Package Navigator view to the UDDI Registry view or by right-clicking a provider web service descriptor in the Package Navigator view and selecting **Publish Web Services to a UDDI Registry**. Designer publishes the service in the UDDI registry with the following characteristics:
  - SOAP 1.1 as the protocol
  - SOAP Document/Literal as the style and use
  - http://host:port as the as the address of the web service
- You can publish a provider web service descriptor to the CentraSite UDDI registry in two different ways:
  - By publishing the package that contains the provider web service descriptor via metadata publishing.
  - By publishing the provider web service descriptor through the UDDI Registry view.

  If you want to publish IS assets in addition to web services, publish the provider web service descriptor by publishing the entire package. If you expect to publish only web services to CentraSite, publish the individual provider web service descriptor through the UDDI Registry view. Typically, users choose one way or the other for publishing web service assets. Software AG does not recommend using a mixture of publishing methods.
- Before publishing a service to a UDDI registry, be sure to create a provider web service descriptor using the IS service as an operation of the web service.

To publish a service to a UDDI registry

1. From UDDI Registry view, connect to the UDDI registry in which you want to publish the web service.

2. In Package Navigator view, copy the provider web service descriptor that contains an operation for the IS service that you want to expose as a web service.

3. In UDDI Registry view, right-click the entity to which you want to publish the web service and select **Publish Web Service**.

   Designer publishes the service to the UDDI registry.

Notes:

- The governance policies established in the registry may delay the display of the published web service in the UDDI Registry view. Refresh the UDDI registry by selecting **Refresh UDDI Registry Display** from the context menu to view the published service.
- If Designer cannot display the service you just published in the UDDI Registry view because of the filter setting, Designer will display a message stating so. Clear the applied filter by selecting **Clear UDDI Filter** from the context menu.
- If you publish the same web service descriptor to a registry twice (without first deleting the original version), two web services will exist with the same name in the registry. However, the **Service Key** property values will be different for these two services. If you cannot determine which version is the newest, delete both web services from the UDDI registry and re-publish the new version.

Deleting a Service from UDDI Registry

Perform the following steps to delete a web service from a UDDI registry. When you remove a web service from a registry, the associated web service descriptor and its supporting elements still exist on the Integration Server but the web service is no longer published in the UDDI registry.

Note:
You cannot delete a web service from another business' folder in the registry. The **Delete** button will be disabled.

To delete a web service from UDDI registry

1. In UDDI Registry view, right-click the web service that you want to remove and select **Delete**.

2. Click **Yes** to confirm deleting the web service.

38 Working with Flat Files

- Concepts 834
- Creating Flat File Schemas 835
- Testing Flat File Schemas 849
- Creating Flat File Dictionaries 852
- Defining Flat File Elements 855

You can translate documents into and from flat file formats using the functionality and services provided in the webMethods Flat File package (WmFlatFile). You can also use these services as templates to create services in Designer that can convert between flat file documents and IS documents (IData objects). The services in the WmFlatFile package also provide a way to manage dictionary entries, entire flat file dictionaries, and flat file schemas.

To set up the translation, you use a flat file schema to define how to identify individual records within a flat file and what data is contained in each of those records. For detailed information about the services in the WmFlatFile package and processing flat files, see *webMethods Integration Server Built-In Services Reference*.

Concepts

You can use the flat file features to translate documents into and from flat file formats. To set up the translation, you use a flat file schema to define how to identify individual records within a flat file and what data is contained in each of those records. You can also create a flat file dictionary to contain the flat file elements (records, composites, fields) that you want to make available for use in all flat file schemas.

What Is a Flat File Schema?

A flat file schema is the blueprint that contains the instructions for parsing or creating a flat file. The schema exists as a namespace element in the webMethods Integration Server. This blueprint details the structure of the document, including delimiters, records, and repeated record structures. A flat file schema also acts as the model against which you can validate an inbound flat file. A flat file schema consists of hierarchical elements that represent each record, field, and subfield in a flat file. Each element is a *record*, *composite*, or *field*, and either a *definition* or *reference*. You configure each element with the necessary constraints. For more detailed information about flat files and flat file schemas, see *Flat File Schema Developer's Guide*.

What Is a Flat File Dictionary?

A flat file schema can contain either record definitions or references to record definitions that are stored elsewhere in the namespace in a *flat file dictionary*. A flat file dictionary is simply a repository for elements that you reference from flat file schemas. This allows you to create record definitions in a dictionary that can be used across multiple flat file schemas. Reusing record definitions reduces the amount of memory consumed by a flat file schema.

Flat file dictionaries are created as namespace elements in Integration Server and contain definitions of records, composites, and fields. When you change a definition in a flat file dictionary that is referenced in multiple flat file schemas, the element definition is updated automatically in all the flat file schemas.

Note:
You can reference a flat file dictionary definition in any flat file schema regardless of whether the dictionary and schema are located in the same package.

When creating an element definition in a flat file dictionary, you specify only certain properties. You then specify the remaining properties in the instance of the element definition in a particular flat file schema.

When Should I Create a Flat File Dictionary?

The decision to define records in a flat file dictionary versus in a flat file schema depends on the type of flat files that you intend to parse. The Electronic Document Interchange (EDI) ANSI X12 standard defines a large set of document structures that reuse the same record, field, and composite definitions many times. Defining these records, fields, and composites in a dictionary allows for them to be reused throughout the entire set of EDI ANSI X12 document flat file schemas. Reusing definitions reduces the amount of memory consumed by Integration Server.

EDI ANSI X12 also has different versions of these documents (for example, 4010). Each version of the document set should have its own dictionary. In this way, you can be certain that changes to a record, field, or composite between versions are maintained.

A more complex scenario would involve multiple families of documents and multiple versions of those families. An example of this is EDI ANSI X12 and UN/EDIFACT documents. One dictionary should be created for each version of EDI ANSI X12 documents and one dictionary should be created for each version of EDI UN/EDIFACT documents. A separate dictionary would not be required for each flat file schema in the same version. All flat file schemas in one version of the same family should use the same dictionary.

In a scenario in which you intend to parse only one flat file, or flat files that do not share record, composite, or field definitions, you can define these elements directly in the flat file schema. This allows for the entire document to be edited in a single view, without referencing a flat file dictionary.

If a clear choice does not exist between these two scenarios, the best approach is to create the definitions in the flat file dictionary and reference them in a flat file schema. The definitions then can be reused at a later time.

Creating Flat File Schemas

You can use Designer to create a flat file schema that Integration Server can use as a blue print for parsing and creating flat files.

Building a flat file schema is a process that involves the following basic stages:

Stage 1	**Create the flat file schema.** During this stage, you create the new flat file schema on the Integration Server where you will do your development and testing. For more information, see "Creating the Flat File Schema" on page 836.
Stage 2	**Define the record parser and specify a record identifier for the flat file schema.** During this stage, you associate a record parser with the flat file schema that will process flat files inbound to the Integration Server. You also specify how you want the record to be identified after it is parsed. For more information about defining the record parser, see "Specifying a Record Parser" on page 837. For more information about specifying a record identifier, see "Specifying a Record Identifier" on page 845.

Stage 3	**Define the structure.**During this stage, you specify the hierarchical structure of the flat file by creating and nesting record definitions or record references. For more instructions, see "Defining the Schema Structure" on page 846.
Stage 4	**Set properties for the flat file schema.**During this stage, you set up the ACL (access control lists) permissions, configure a default record, add areas, and allow undefined data for your flat file schema or dictionary.
Stage 5	**Test the flat file schema.**During this stage, you can use the tools provided by Designer to test the flat file schema. For more information, see "Testing a Flat File Schema" on page 851.

Note:
When validation is enabled, Integration Server can generate errors for the **Ordered**, **Mandatory**, **Validator**, and **Undefined Data** properties. To enable validation, you must set the validate variable of the convertToValues service to `True`. For more information about this service, see *webMethods Integration Server Built-In Services Reference*.

Creating the Flat File Schema

When you create a flat file schema, keep the following points in mind:

- You must have Write access to the folder in which you want to store the flat file schema.
- You can quickly create a flat file schema by right-clicking the folder, selecting **New > Flat File Schema**. Enter a name for the flat file schema in the Provide a Name dialog box and click **OK**. Designer automatically creates a flat file schema in the selected folder.

To create a flat file schema

1. In the Package Navigator view of Designer, click **File > New > Flat File Schema**.
2. In the New Flat File Schema wizard, select the folder in which you want to save the flat file schema.
3. In the **Element name** field, type a name for the flat file schema using any combination of letters, numbers, and/or the underscore character. For information about restricted characters, see "About Element Names" on page 41.
4. Click **Finish**.

   Integration Server generates a flat file schema and Designer displays it in the Package Navigator view.
5. Next, use the flat file schema editor to configure the record parser and record identifier. See "Specifying a Record Parser" on page 837.

Specifying a Record Parser

Integration Server can exchange all types of flat files but can process only certain types of flat files. Integration Server can process flat files in which:

- The records in the flat file are defined using one of the following methods:
  - **Delimiter.** Each record in the flat file is separated by a delimiter.
  - **Fixed length.** Each record is a fixed number of bytes (for example, mainframe punch or print records).
  - **Variable length.** Each record is preceded by two bytes that indicate the length of the record. Records in the flat file can have different lengths.
  - **EDI document type.** This option is only for viewing existing EDI flat file schemas.

    Note:
    If you are using the webMethods Module for EDI to process EDI documents, you should use the wm.b2b.edi services to create your flat file schemas. This help system does not provide information about creating EDI flat file schemas for use with the webMethods Module for EDI. For more information and steps, see the *webMethods Module for EDIINT Installation and User's Guide* . The **EDI Document Type** option is displayed for you to view existing EDI flat file schemas.
- Each distinct record structure has a record identifier. If no record identifier is present, the record is parsed using a default record definition. For more information about default records, see "Setting a Default Record" on page 846.
- If the flat file contains record identifiers, the record identifiers must be located in the same location in all records in the file.

Integration Server then can identify fields in these records based on either:

- **Delimiters.** Each field is separated by a delimiter, and you can specify the *N*th delimited field in a record to represent the record identifier. This can be used only when a field delimiter (and, if necessary, subfield delimiter) has been specified.
- **Byte position.** Each field is defined by 1) the number of bytes from the beginning of the record and 2) the field length. This can be used regardless of whether a field delimiter has been specified.

Specifying a Delimited Record Parser for the Schema

Use this parser when each record is separated by a delimiter. For a record delimiter, you can specify:

- A character (for example, !) or character representation (for example, `\r\n` for carriage return).
- Hexadecimal value (for example, 0X09).
- Octal value (for example, 009).

- Unicode characters (for example, \uXXXX, where XXXX represents the Unicode value of the character).

To configure a delimited record parser

1. In Package Navigator view, double-click the flat file schema that you want to configure. The flat file schema opens in the flat file schema editor window.

2. In the **Record Parse Type** area, select **Delimited**.

3. Specify the following fields:

   a. **Record**

Property	Description
Character	Character that separates records in a flat file document.
--OR--	
Character Position	Starting from the beginning of the document and counting from zero (0), the character position at which the record delimiter for this document is located. For example, if you specify 3 as the character position, you have indicated that the record delimiter appears in the fourth character position from the beginning of the document. **Note:** If records use a fixed position extractor, the delimiter record parser does not include the record delimiter character in the parsed record. You will not be able to extract the record delimiter character in a fixed position field.

   b. **Field or composite**

Property	Description
Character	Optional. Character that separates fields in a flat file document.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the field delimiter for this document is located. For example, if you specify 4 as the character position, you have indicated that the field delimiter appears in the

Property	Description
	fifth character position from the beginning of the document.

c. **Subfield**

Property	Description
Character	Optional. Character that separates subfields in a flat file document. The default is a period ".".
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the subfield delimiter for this document is located. For example, if you specify 5 as the character position, you have indicated that the subfield delimiter appears in the sixth character position from the beginning of the document.

d. **Quoted release character**

Property	Description
Character	Optional. Character used to enable a section of text within a field to be represented as its literal value. Any delimiter characters that appear within this section will not be treated as delimiters. For example, your field delimiter is (,) and your release character is ". When you want to use (,) within a field as text, you must preface it with your quoted release character. When using the convertToValues service to create the strings `Doe, John` and `Doe, Jane`, the record would appear as "`Doe, John`","`Doe, Jane`". When using the convertToString service to create "`Doe, John`","`Doe, Jane`", the value of the record would be `Doe, John and Doe, Jane`. When using the convertToString service, if you have specified both the Release Character and the Quoted Release Character, the Quoted Release Character will be used.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the quoted release character for this document is located. For example, if you specify 5 as the character

Property	Description
	position, you have indicated that the quoted release character appears in the sixth character position from the beginning of the document.

e. **Release character**

Property	Description
Character	Optional. Character used to enable a delimiter to be used for its intended, original meaning. The character following the release character will not be treated as a delimiter. For example, your field delimiter is + and your release character is \. When using + within a field as text, you must preface it with your release character. When using the convertToValues service to create the strings a+b+c and d+e+f, the record would appear as a\+b\+c+d\+e\+f. When using the convertToString service to create a\+b\+c+d\+e\+f, the value of the record would be a+b+c and d+e+f.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the field delimiter for this document is located.For example, if you specify 5 as the character position, you have indicated that the field delimiter appears in the sixth character position from the beginning of the document.

4. Next, set the record identifier for the schema. See "Specifying a Record Identifier" on page 845.

Specifying a Fixed Length Record Parser for the Schema

Use a fixed length record parser when each record is of a fixed length (for example, mainframe punch or print records). This parser splits a file into records of the same pre-specified length.

To configure a fixed length record parser

1. In Package Navigator view, double-click the flat file schema that you want to configure. The flat file schema opens in the flat file schema editor window.

2. In the **Record Parse Type** area, select **Fixed Length**.

3. In the **Record Length** field, enter the length, in characters, of each record in the flat file.

4. Specify the following fields:

   a. **Field or composite**

Property	Description
Character	Optional. Character that separates fields or composites in a flat file document.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the field delimiter for this document is located. For example, if you specify 4 as the character position, you have indicated that the field delimiter appears in the fifth character position from the beginning of the document.

   b. **Subfield**

Property	Description
Character	Optional. Character that separates subfields in a flat file document.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the subfield delimiter for this document is located. For example, if you specify 5 as the character position, you have indicated that the subfield delimiter appears in the sixth character position from the beginning of the document.

   c. **Quoted release character**

Property	Description
Character	Optional. Character used to enable a section of text within a field to be represented as its literal value. Any delimiter characters that appear within this section will not be treated as delimiters. For example, your field delimiter is (,) and your release character is ". When you want to use (,) within a field as text, you must preface it with your quoted release

Property	Description
	character. When using the convertToValues service to create the strings `Doe, John` and `Doe, Jane`, the record would appear as "`Doe, John`","`Doe, Jane`". When using the convertToString service to create "`Doe, John`","`Doe, Jane`", the value of the record would be `Doe, John and Doe, Jane`.When using the convertToString service, if you have specified both the Release Character and the and the Quoted Release Character, the Quoted Release Character will be used.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the quoted release character for this document is located. For example, if you specify 5 as the character position, you have indicated that the quoted release character appears in the sixth character position from the beginning of the document.

d. **Release character**

Property	Description
Character	Optional. Character used to enable a delimiter to be used for its intended, original meaning. The character following the release character will not be treated as a delimiter. For example, your field delimiter is + and your release character is \. When using + within a field as text, you must preface it with your release character. When using the convertToValues service to create the strings a+b+c and d+e+f, the record would appear as a\+b\+c+d\+e\+f. When using the convertToString service to create a\+b\+c+d\+e\+f, the value of the record would be a+b+c and d+e+f.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the field delimiter for this document is located. For example, if you specify 5 as the character position, you have indicated that the field delimiter appears in the sixth character position from the beginning of the document.

5. Next, set the record identifier for the schema. See "Specifying a Record Identifier" on page 845.

Specifying a Variable Length Record Parser for the Schema

This parser expects each record to be preceded by two bytes that indicate the length of the record. Each record may be a different length.

To configure a variable length record parser

1. In Package Navigator view, double-click the flat file schema that you want to configure. The flat file schema opens in the flat file schema editor window.

2. In the **Record Parse Type** area, select **Variable length**.

3. Specify the following fields:

   a. **Field or composite**

Property	Description
Character	Optional. Character that separates fields or composites in a flat file document.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the field delimiter for this document is located. For example, if you specify 4 as the character position, you have indicated that the field delimiter appears in the fifth character position from the beginning of the document.

   b. **Subfield**

Property	Description
Character	Optional. Character that separates subfields in a flat file document.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the subfield delimiter for this document is located.For example, if you specify 5 as the character position, you have indicated that the subfield delimiter appears in the sixth character position from the beginning of the document.

c. **Quoted release character**

Property	Description
Character	Optional. Character used to enable a section of text within a field to be represented as its literal value. Any delimiter characters that appear within this section will not be treated as delimiters.For example, your field delimiter is (,) and your release character is ". When you want to use (,) within a field as text, you must preface it with your quoted release character. When using the convertToValues service to create the strings `Doe, John` and `Doe, Jane`, the record would appear as "`Doe, John`","`Doe, Jane`". When using the convertToString service to create "`Doe, John`","`Doe, Jane`", the value of the record would be `Doe, John and Doe, Jane`.When using the convertToString service, if you have specified both the Release Character and the and the Quoted Release Character, the Quoted Release Character will be used.
--OR--	
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at which the quoted release character for this document is located. For example, if you specify 5 as the character position, you have indicated that the quoted release character appears in the sixth character position from the beginning of the document.

d. **Release character**

Property	Description
Character	Optional. Character used to enable a delimiter to be used for its intended, original meaning. The character following the release character will not be treated as a delimiter. For example, your field delimiter is + and your release character is \. When using + within a field as text, you must preface it with your release character. When using the convertToValues service to create the strings a+b+c and d+e+f, the record would appear as a\+b\+c+d\+e\+f. When using the convertToString service to create a\+b\+c+d\+e\+f, the value of the record would be a+b+c and d+e+f.
Character position	Optional. Starting from the beginning of the document and counting from zero (0), the character position at

Property	Description
	which the field delimiter for this document is located. For example, if you specify 5 as the character position, you have indicated that the field delimiter appears in the sixth character position from the beginning of the document.

4. Next, set the record identifier for the schema. See"Specifying a Record Identifier" on page 845.

Specifying a Record Identifier

When parsing a file, Integration Server looks at a record and extracts an identifier out of the data. The server uses that identifier to connect the record definition in a flat file schema with a particular record in the flat file. The name of the record definition must match the value obtained by the record identifier. You can choose from one of two methods of record identification:

- **Starts at position** record identifiers compare the value that occurs in the record, at the specified offset, to all the record names defined in the flat file schema. Note that the **Starts at** position identifier cannot distinguish between all types of record names. For example, if you name records "Rec1" and "Rec," some instances of "Rec1" may be identified as "Rec," because "Rec1" begins with "Rec."
- **Nth Field record** identifiers use the value of the specified field as the record identifier. These identifiers count from zero (0). For example, if 2 is specified, the third field is used as the record identifier.

To set the record identifier for a schema definition

1. In Package Navigator view, double-click the flat file schema that you want to configure. The flat file schema opens in the flat file schema editor window.
2. In the **Record identifier** area, set the record identifier to one of the following values:

Value	Description
Starts at position	Identifies the character position in the record (counting from zero) where the record identifier is located.
NthField	Identifies the field in the record (counting from zero) that contains the identifier.

3. Next, define the structure for the flat file schema. For instructions, see "Defining the Schema Structure" on page 846.

Defining the Schema Structure

The application receiving the flat file uses the structure that is defined in the flat file schema to read the flat file. This structural information identifies the parent-child relationships between different records in the flat file. By nesting record elements in the flat file schema (adding record elements to a record) you can represent the hierarchical structure of the data in the flat file.

Use the **Structure** tab in the flat file schema editor to add records to the flat file schema and to define the hierarchical relationships between them.

To define the flat file schema

1. In Package Navigator view, double-click the flat file schema to which you want to add an element. The schema opens in the flat file schema editor window.

2. To add the first record, click the name of the flat file schema in the Name column, and then click . (You can also right-click an element and select **New**.)

3. Select one of the following elements:

Element	See...
Record Definition	"Adding a Record Definition" on page 856
Record Reference	"Adding a Record Reference" on page 856

4. Add flat file elements to define the structure of the flat file schema. You can add additional records. You can also further define records by adding child composite and field definitions. For instructions about adding, configuring, and nesting flat file elements in your flat file schema, see "Defining Flat File Elements" on page 855.

Setting a Default Record

You can select a default record from a flat file dictionary when creating a flat file schema. This record is used to parse an undefined data record when the convertToValues service fails to find a match between the flat file and the flat file schema. In other words, Integration Server will use the default record to parse any records it does not recognize.

Keep the following in mind when deciding whether or not to set a default record:

- If your flat file does not contain record identifiers, you must select a default record. By selecting a default record, a CSV (comma separated values) file can be parsed as a special case of record with no record identifier, but with fixed field and record delimiters.
- If you do not select a default record, unrecognized records are placed into the output IS document with the undefined data tag, which might produce errors.

- If the default record is specified when creating the flat file schema, any record that cannot be recognized will be parsed using this default record. If a default record is not selected, the record will be treated as undefined data. If the **Undefined Data** property is set to `False` and the validate variable of the convertToValues service is set to true, convertToValues will generate errors when it encounters undefined data. For more information about the **Undefined Data** property, see "Allowing Undefined Data" on page 847

To specify a default record

1. In the Package Navigator view of Designer, open the flat file schema to which you want to add a default record.
2. In the Properties view under the **Default Record** area, click … next to **Set**.
3. Browse to and select the default record for this flat file schema from a flat file dictionary, then click **Next**.
4. Select the record definition to use as the default record, then click **Finish**.
5. Save the flat file schema.

Allowing Undefined Data

You can configure how Integration Server handles undefined data errors that are generated at any location in the flat file schema.

When the convertToValues service processes an undefined record, it puts a placeholder named `unDefData` in the resulting IS document and stores the record as a string in the pipeline.

Note:
If the file is encoded using a multi-byte encoding, and if you use a fixed length or variable length parser, the service puts two placeholders into the pipeline: `unDefData` and `unDefBytes`.

To specify how Integration Server handles undefined data

1. In the Package Navigator view of Designer, open the flat file schema to which you want to specify how you want Integration Server to handle undefined data.
2. In the Properties view next to **Allow undefined data**, choose one of the following options:

Select...	To...
True	Allow undefined data. If you select this option, you can choose whether to allow undefined data at the record level.

Select...	To...
False	Not allow undefined data in any location in this flat file schema. This is the default. If the validate variable of the convertToValues service is set to `true`, the convertToValues service will generate errors when undefined data is encountered.

3. Save the flat file schema.

Creating an Area

An area is a way to associate an arbitrary string with a given record. For example, you may have an address record that needs to specify the shipping address in the document header, but needs to specify the billing address in the document detail. To differentiate these two records, you would create "Header" and "Detail" areas.

Note:
Areas are used primarily for EDI document parsing.

To create an area

1. In the Package Navigator view of Designer, open the flat file schema to which you want to add an area.

2. In the Properties view in the **Settings** area, click … next to **Areas**.

3. Click to add a new area to the flat file schema. Click to insert a new area in a specific location in the schema. Click to delete an existing area.

4. Save the flat file schema.

Specifying a Floating Record

Use the **Floating Record** property to designate any single record of a given flat file schema to be a floating record. By designating a floating record, you enable that record to appear in any position within a flat file without causing a parsing validation error.

Note:
If you do not use this property, validation errors will occur if the record structure of an inbound document does not match the record structure defined in its flat file schema.

To specify a floating record

1. In the Package Navigator view of Designer, open the flat file schema to which you want to add a floating record.

2. In the Properties view next to **Floating Record**, click ....

3. Enter the name of the existing record that you want to designate as the floating record, then click **OK**. If a floating record has an alternate name, specify the alternate name in this field. For more information about alternate names, see "Record Definition Properties" on page 971.

4. Save the flat file schema.

Editing a Flat File Schema

To edit a flat file schema, you must have the proper access permissions to do so and have locked the flat file schema. For information about access control lists (ACLs) see "Assigning and Managing Permissions for Elements" on page 79.

Editing a flat file dictionary is the same as editing a flat file schema except there is no **Structure** tab in the dictionary editor.

You can do the following:

- **Edit any element properties**. To do so, make the necessary changes on the **Flat File Structure** tab and then save the flat file schema. For a list of these settings and properties, see "Creating the Flat File Schema" on page 836.
- **Rename existing elements.** To do so, right–click the element, and then select **Rename**.
- **Re–structure the elements.** To rearrange the structure of your flat file schema, use the buttons at the top of the **Flat File Structure** tab.

Button	Description
	Select the element you want to delete, and click to delete.
	Select the element you want to move, and click to move the element up or down in the flat file schema structure.
	Select the element you want to move, and click to move the element left or right in the flat file schema structure.

Testing Flat File Schemas

You can test a flat file schema to verify that the parsing information and structure defined for the flat file schema result in correctly parsed and processed flat file documents. To test a flat file schema, create a launch configuration for the flat file schema. The launch configuration specifies the encoding, whitespace handling, and flat file to use when testing the flat file schema. Designer uses this information to parse and process the supplied flat file using the flat file schema. Designer displays the test results in the Results view.

Creating a Launch Configuration for a Flat File Schema

Use the following procedure to create a launch configuration for testing a flat file schema.

To create a launch configuration for a flat file schema

1. In the Service Development perspective, select **Run > Run Configurations**.
2. In the Run Configurations dialog box, select **Flat File Schema** and click .
3. In the **Name** field, specify a name for the new launch configuration.
4. On the **Flat File** tab, in the **Integration Server** list, select the Integration Server on which the flat file schema for which you want to create a launch configuration resides.
5. In the **Flat File Schema** field, click **Browse** to navigate to and select the flat file schema, then click **OK**.

   Tip:
   If you select the flat file schema in Package Navigator view and then select **Run > Run Configurations**, Designer populates the **Integration Server** and **Flat File Schema** fields automatically.
6. On the **Input** tab, in the **Skip whitespace** list, select true if you want Designer to ignore whitespace at the beginning of a record.

   Note:
   If the flat file schema specifies a fixed length parser, Designer always preserves whitespace when processing a flat file document. For fixed length parsers, the **Skip whitespace** value is ignored.
7. In the **Encoding** list, select the encoding for the flat file that you will be testing.
8. Next to the **File** field, click the **Browse** button to navigate to and select the flat file that you want this launch configuration to use when testing the flat file schema.
9. Optionally, click the **Common** tab to specify general information about the launch configuration and to save the launch configuration to a file.
10. Click **Apply**.
11. Click **Run** to test the flat file schema now. Otherwise, click **Close**.

Testing a Flat File Schema

By testing a flat file schema, you can discover any errors in the flat file schema that prevent flat files from being parsed correctly. Keep the following points in mind when testing a flat file schema:

- When you test a flat file schema, you can select the launch configuration that Designer uses. If a launch configuration does not exist for the flat file schema, Designer creates a launch configuration and immediately prompts you for input values. Designer then runs the launch configuration. Designer saves the launch configuration in your workspace
- Designer always performs validation when testing the flat file schema. To enable validation when processing a flat fie document using the pub.flatFile:convertToValues service, set the *validate* input parameter to true.
- If the flat file schema specifies a fixed length parser, Designer always preserves whitespace when processing a flat file document. For fixed length parsers, the **Skip whitespace** value is ignored.

To test a flat file schema

1. In the Service Development perspective, in Package Navigator view, select the flat file schema to test.
2. Select **Run > Run As > Flat File Schema**.
3. If multiple launch configurations exist for the flat file schema, in the **Select Launch Configurations** dialog box, select the launch configuration that you want Designer to use and click **OK**.
4. If the launch configuration is set up to prompt for input before running or there is no launch configuration for the flat file schema, Designer displays the **Enter Input for** *`flatFileSchemaName`* dialog box. If the launch configuration is not configured to prompt for input, Designer runs the launch configuration.
5. To test the flat file schema using the input data provided in the launch configuration, click **OK** to run the launch configuration. Otherwise, specify the following input data:
    a. In the **Skip whitespace** list, select true if you want Designer to ignore whitespace at the beginning of a record.
    b. In the **Encoding** list, select the encoding for the flat file that you will be testing.
    c. Next to the **File** field, click the **Browse** button to navigate to and select the flat file that you want this launch configuration to use when testing the flat file schema.
    d. Click **OK** to run the launch configuration.

Designer displays the results in the Results view. The Pipeline tab contains the document (IData) created by parsing and processing the supplied flat file. The Message tab contains any errors encountered during parsing and processing.

Creating Flat File Dictionaries

You can use Designer to create a flat file dictionary that contains definitions of records, composites, and fields that can be used across multiple flat file schemas.

Building a flat file dictionary is a process that involves the following basic stages:

Stage 1	**Create the flat file dictionary.** During this stage, you create the new flat file dictionary on Integration Server. For more information, see "Creating a Flat File Dictionary" on page 852.
Stage 2	**Add Elements to the Flat File Dictionary.** During this stage, you add elements to the Record Definition, Composite Definition, or Field Definition elements of the flat file dictionary. For more information, see "Adding Elements to the Flat File Dictionary" on page 853.
Stage 3	**Set Properties for the Flat File Dictionary.** During this stage, you set up the ACL (access control lists) permissions, configure the default record, allow undefined data, and specify floating records for your flat file dictionary. For more information, see "Setting Properties for the Flat File Dictionary" on page 853.

Creating a Flat File Dictionary

You can create a flat file dictionary for use with flat file schemas. A flat file dictionary contains definitions of records, composites, and fields that can be used across multiple flat file schemas. Reusing these definitions reduces the amount of memory consumed by a flat file schema. Flat file dictionaries can also contain references to composite and field definitions in this dictionary and/or in other dictionaries.

For more information about the circumstances under which you might create and use a flat file dictionary, see "When Should I Create a Flat File Dictionary?" on page 835.

To create a flat file dictionary

1. In the Package Navigator view of Designer, click **File > New > Flat File Dictionary**.

2. In the New Flat File Dictionary dialog box, type the name of the new flat file dictionary. Select a folder in which to save the flat file dictionary. You must have Write access to the folder in which you want to save the flat file dictionary and this folder must be dependent on the WmFlatFile package.

   For more information about package dependencies, see "About Package Dependencies" on page 149.

3. Click **Finish**.

   The flat file dictionary you created is automatically selected, and Designer displays it in the Package Navigator view. You now can use the flat file dictionary editor to configure the flat file dictionary.

   Note:
   You can quickly create a flat file dictionary by right-clicking the folder, selecting **New > Flat File Dictionary**. Enter a name for the flat file dictionary in the New Flat File Dictionary dialog box and click **Finish**. Designer automatically creates a flat file dictionary in the selected folder.

Adding Elements to the Flat File Dictionary

You can add elements to the Record Definition, Composite Definition, or Field Definition elements of the flat file dictionary.

To add elements to the flat file dictionary

1. In the Package Navigator view of Designer, open the flat file dictionary to which you want to add elements.

2. Select the element type to which you want to add an element (**Record Definition**, **Composite Definition**, or **Field Definition**) and click in the flat file dictionary editor toolbar.

   The New Flat File Element dialog box displays the valid elements that you can add to the element type you have selected. For more information about adding elements to flat file dictionary, see"Defining Flat File Elements" on page 855.

3. After you have added all the elements to the flat file dictionary, save the dictionary. You now can create flat file schemas based on this flat file dictionary.

   Note:
   You cannot create references to the elements added to a dictionary until you save the dictionary.

Setting Properties for the Flat File Dictionary

You use the Properties view of the flat file dictionary editor to set up the ACL (access control lists) permissions, configure a default record, allow undefined data, and specify floating records for your flat file dictionary.

To set properties for the flat file dictionary

1. In the Package Navigator, open the flat file dictionary for which you want to set the properties. The flat file dictionary opens in the flat file dictionary editor window.

2. In the Properties view of the flat file dictionary editor, specify the properties for the selected record.

   For each of the following element levels, you can specify the following properties in the Properties view:

Element	Property
Record Definition	■ Validator ■ Check Fields ■ Alternate Name ■ Description To define other properties, you must do so in the flat file schema. For a list of all properties, see "Composite Definition Properties" on page 975.
Composite Definition	■ Validator ■ Check Fields ■ Alternate Name ■ Description ■ ID Code To define other properties, you must do so in the flat file schema. For a list of all properties, see "Composite Definition Properties" on page 975.
Field Definition	■ Validator ■ Format Service ■ Alternate Name ■ Description ■ ID Code ■ DataType To define other properties, you must do so in the flat file schema. For a list of all properties, see "Field Definition Properties" on page 980.

3. After you have specified the properties for the selected record, save the dictionary. You now can create flat file schemas based on this flat file dictionary.

Editing a Flat File Dictionary

You can edit, rename, or delete any element in a flat file dictionary from the dictionary editor. Editing a flat file dictionary is very similar to editing a flat file schema. For more information about editing, see "Editing a Flat File Schema" on page 849.

Note:
To edit a flat file dictionary, you must have the proper access permissions and must lock the flat file dictionary. For information about access permissions see "Assigning and Managing Permissions for Elements" on page 79

Defining Flat File Elements

This topic applies to both flat file schemas and flat file dictionaries, unless otherwise noted.

To add elements to a flat file schema you must be in the Flat File Structure tab of the flat file schema editor.

To add elements to a flat file dictionary you must be in the flat file dictionary editor.

To add elements to a flat file schema or flat file dictionary

1. In Package Navigator view, double-click the flat file schema or flat file dictionary to which you want to add an element. The schema or dictionary opens in the appropriate editor window.

2. In the Flat File Structure tab of the flat file schema editor, or in the flat file dictionary editor, select the element to which you want to add an element and click the in the editor tool bar. (You can also right-click an element and select **New**.)

3. Select the element to which you want to add an element.

4. The New Flat File Element dialog box displays the valid elements that you can add to the element type you have selected.

Element	See...
Record Definition	"Adding a Record Definition" on page 856
Record Reference	"Adding a Record Reference" on page 856
Composite Definition	"Adding a Composite Definition" on page 857
Composite Reference	"Adding a Composite Reference" on page 858
Field Definition	"Adding a Field Definition" on page 858
Field Reference	"Adding a Field Reference" on page 860

Adding a Record Definition

To add a record definition

1. In Package Navigator view, double-click the flat file schema or flat file dictionary to which you want to add a record definition. The schema or dictionary opens in the appropriate editor window.

2. In the Flat File Structure tab of the flat file schema editor, or in the flat file dictionary editor, select the schema or dictionary and click in the editor tool bar. (You can also right–click and select **New**.)

3. Select **Record Definition** and click **Next**.

4. Specify a name for the record definition in the Enter Record Definition Name dialog box and click **Finish**.

Adding a Record Reference

For flat file schemas only.

In a flat file schema, you can create a reference to a record definition that is defined in a flat file dictionary.

To add a record reference

1. In Package Navigator view, double-click the flat file schema to which you want to add a record reference. The flat file schema opens in the flat file schema editor window.

2. In the Flat File Structure tab of the flat file schema editor, select the schema and click in the editor toolbar. (You can also right–click and select **New**.)

3. Select **Record Reference** and click **Next**.

4. Navigate to the flat file dictionary in which the record is located, select the dictionary, and then click **Next**.

5. Select the element that you want to reference and then click **Next**.

6. In the Enter Record Definition Name(s) dialog box, type the name of the record.

   Important:
   This name must match the value of its record identifier exactly as it will appear in the flat file. The name of a record reference does not have to match the name of the record definition in the flat file dictionary. The name of a record reference will be matched to the record

identifier in the record. The name of the record definition in the flat file dictionary does not need to match the record identifier that appears in the flat file.

7. Click **Finish**.

   The record is added to the flat file schema structure. The **Referring To** field indicates the record definition to which the record reference refers. The **Dictionary** field indicates the flat file dictionary to which the record reference refers. If the element is a record definition, these two fields are empty.

Adding a Composite Definition

To add composite definition

1. In Package Navigator view, double-click the flat file schema or flat file dictionary to which you want to add a composite definition. The schema or dictionary opens in the appropriate editor window.

2. In the Flat File Structure tab of the flat file schema editor, or in the flat file dictionary editor, select a record definition or composite definition and click in the editor toolbar. (You can also right-click the element and select **New**.)

3. Select **Composite Definition** and click **Next**.

4. Under **Enter Composite Definition Information**, specify the following:

Property	Description
Name	Name for the composite definition.
Position	Field number in the record that contains the composite you want to extract. This pulls the subfield data from the composite. If you leave this property empty, the composite will not be extracted.
Mandatory	Optional. Select the check box to require that this composite appear in the flat file. If it is not selected, the composite is not required to appear in the flat file. If it is selected and the convertToValues service validate variable is set to true, errors will be generated if the composite does not appear in the flat file.

5. To add additional composite definitions at this level, click .

6. Click **Finish** to save the element definition.

Adding a Composite Reference

From a schema, you can add a composite reference that points to a dictionary. From a dictionary, you can add a composite reference that points to the current dictionary, or to another dictionary.

To add a composite reference

1. In Package Navigator view, double-click the flat file schema or flat file dictionary to which you want to add a composite reference. The schema or dictionary opens in the appropriate editor window.

2. In the Flat File Structure tab of the flat file schema editor, or in the flat file dictionary editor, select the record definition and click in the editor toolbar. (You can also right-click the element and select **New**.)

3. Select **Composite Reference** and click **Next**.

4. Navigate to the flat file dictionary in which the element is located, select the dictionary, and then click **Next**.

5. Select the element that you want to reference and then click **Next**.

6. Enter the details required in the **Enter Composite Reference Name(s)** as specified in "Adding a Composite Definition" on page 857.

7. Click **Finish**.

Adding a Field Definition

To add a field definition

1. In Package Navigator view, double-click the flat file schema or flat file dictionary to which you want to add a field definition. The schema or dictionary opens in the appropriate editor window.

2. In the Flat File Structure tab of the flat file schema editor, or in the flat file dictionary editor, select the record definition or composite definition and click in the editor toolbar. (You can also right-click the element and select **New**.)

3. Under **Enter Field Definition Information**, specify the following.

Extractor Type	Description
Fixed Position	Counting from zero (0), indicates a fixed number of bytes to be extracted from a record.

Property	Description
Name	Type the name of the field.
Start	Type the first byte to extract from the record.
End	Type the first byte that is not included in the extraction. If you enter a negative number (for example, –1), the extractor returns all bytes from the byte specified in **Start** to the last byte in the record or composite.
Mandatory	Optional. Select the check box to require that this field appear in the flat file. If is not selected, the field is not required to appear in the flat file. If it is selected and the convertToValues service *validate* variable is set to true, errors will be generated if the field does not appear in the flat file.

Nth Field	Counting from zero (0), indicates the field that you want to extract from the record.

Property	Description
Name	Type the name of the field.
Position	Type a value to indicate the position of the field that you want to extract from the record. This value cannot be null and must be an integer greater than or equal to zero (0). For example, if you type 1, the second field will be extracted. This option is available only if you specified a field delimiter when configuring your flat file schema. This extractor returns the field as a key–value pair. The key is the name of the field. The value is the String value of the field.
Mandatory	Optional. Select the check box to require that this field appear in the flat file. If is not selected, the field is not required to appear in the flat file. If it is selected and the convertToValues service *validate* variable is set to true, errors will be generated if the field does not appear in the flat file.

4. To add additional field definitions at this level, click .

5. Click **Finish** to save the element definition.

Adding a Field Reference

From a schema, you can add a field reference that points to a dictionary. From a dictionary, you can add a field reference that points to the current dictionary, or to another dictionary.

To add a field reference

1. In Package Navigator view, double-click the flat file schema or flat file dictionary to which you want to add a field reference. The schema or dictionary opens in the appropriate editor window.

2. In the Flat File Structure tab of the flat file schema editor, or in the flat file dictionary editor, select the record definition or composite definition and click in the editor toolbar.

3. Select **Field Reference** and click **Next**.

4. Navigate to the flat file dictionary in which the element is located, select the dictionary, and then click **Next**.

5. Select the element that you want to reference and then click **Next**.

6. Enter the details required in the **Enter Field Reference Information** as specified in "Adding a Field Definition" on page 858.

7. Click **Finish**.

39 Working with Adapters

- About Adapter Connections 862
- About Adapter Services 862
- About Adapter Listeners 862
- About Adapter Notifications 863

webMethods adapters connect resources in your enterprise to the webMethods product suite and, through the suite, to each other. While Integration Server supports a variety of standards such as XML, adapters support proprietary protocols for accessing packaged applications such as SAP, Siebel, JD Edwards, Oracle Applications, and PeopleSoft; databases such as Oracle, SQL Server, Informix, Sybase, and DB2; and mission-critical programs on mainframes and UNIX systems. Adapters transform data from resource-specific format into the format used within the webMethods product suite, and vice versa. By using adapters you can incorporate resources into integration solutions without having to build complex custom code. Adapters run on Integration Server.

Adapters convey data from resources to the webMethods product suite. Adapters can either actively poll resources for new or changed data or passively receive new or changed data from resources. For example, the webMethods Adapter for JDBC can receive data from a database, transform it from the database-specific format into the webMethods format, and send the transformed data to services on Integration Server for further processing.

Adapters also convey data from the webMethods product suite to resources. For example, a JDBC Adapter service can receive data from an Integration Server service, transform it from webMethods format into the format required by the database, and insert it into the database.

About Adapter Connections

An adapter connection is an object that contains parameters that adapter notifications and listeners use to connect to an adapter resource. When you create an adapter service, adapter polling notification, or listener, you specify the adapter connection that the service or polling notification uses to connect to the resource.

You use Integration Server Administrator to create and manage adapter connections. In Designer, you can view information about an adapter connection by clicking the adapter connection in Package Navigator view. For information about creating adapter connections, see the documentation provided with the adapter.

About Adapter Services

An adapter service defines an operation that the adapter will perform on an adapter resource. Adapter services operate like flow services or Java services. Adapter services have an input and output signature, can be invoked within a flow service, can be used as an operation in a provider web service descriptor, and can generate audit logging data. Designer provides facilities to create, configure, and run adapter services.

Each adapter comes with its own unique set of templates for use in creating adapter services. For information about creating adapter services, see the documentation provided with the adapter.

About Adapter Listeners

An adapter listener is an object that uses an adapter connection to connect to an adapter resource and wait for the resource to deliver data when an event occurs on the resource. Listeners work with listener notifications to detect and process event data on the adapter resource. When you create an adapter listener notification, you specify the adapter listener that the adapter uses to connect to the adapter resource.

You use Integration Server Administrator to create and manage adapter listeners. In Designer, you can view information about an adapter listener by clicking the adapter listener in Package Navigator view. For information about creating adapter listeners, see the documentation provided with the adapter.

About Adapter Notifications

An adapter notification contains information about an event that occurs on an adapter resource and then sends the notification data to Integration Server in the form of a published document. There are two types of adapter notifications:

- Polling notifications poll the resource for events that occur on the resource.

  When you create a polling notification in Designer, you specify the notification template and the adapter connection to use to connect to the adapter resource.

- Listener notifications work with listeners to detect and process events that occur on the adapter resource.

  When you create a listener notification in Designer, you specify the notification template Andes listener to use to connect to the adapter resource.

When creating an adapter notification, Designer also creates a publishable document type that describes the data generated by the adapter notification. At run time, the notification publishes this document and sends it to Integration Server.

To process a document associated with an adapter notification, create a webMethods messaging trigger to subscribe to the document type created for the notification. When Integration Server receives a notification document, the subscribing trigger processes the document by invoking the trigger service associated with the document type subscription.

Each adapter comes with its own unique set of templates for use in creating adapter notifications. For information about creating adapter notifications, see the documentation provided with the adapter.

40 Subscribing to Events

- What Happens When an Event Occurs? 867
- Subscribing to Events 868
- Viewing and Editing Event Subscriptions 873
- Suspending Event Subscriptions 873
- Deleting an Event Subscription 874
- Building an Event Handler 874
- Invoking Event Handlers Synchronously or Asynchronously 875
- About Alarm Events 875
- About Audit Events 876
- About Audit Error Events 876
- About Exception Events 876
- About Guaranteed Delivery Events 877
- About JMS Delivery Failure Events 878
- About JMS Retrieval Failure Events 878
- About Port Status Events 879
- About Replication Events 879
- About Security Events 880
- About Session Events 881
- About Stat Events 881

■ About Transaction Events 881

The Event Manager monitors Integration Server for *events* and invokes event handlers when those events occur. An event is a specific action that the Event Manager recognizes and an event handler can react to. An *event handler* is a service that you write to perform some action when a particular event occurs. You then *subscribe* the event handlers to the events about which they need to be notified.

You can use the Event Manager to manage all of your event subscriptions and perform the following tasks:

- Subscribe event handlers to events.
- View or edit event subscriptions.
- Suspend event subscriptions.
- Delete event subscriptions.

Note:
You can also use built-in services to add, modify, and delete event subscriptions. These services are located in the pub.event folder. For more information about built-in services, see the *webMethods Integration Server Built-In Services Reference*.

Note:
The Event Manager monitors local Integration Server events only. It does not monitor EDA (Event Driven Architecture) events.

What Happens When an Event Occurs?

When an event occurs, the Event Manager automatically invokes all event handlers that subscribe to the event. The event handlers receive an input object containing run-time information. The exact content of this input object varies depending on the type of event that occurred and, for audit events, the run-time properties set on both Integration Server and the service that generated the event.

Other points to keep in mind about events and event handlers:

- An event can have more than one subscriber, which means that a single event might invoke several event handlers.
- If an event invokes more than one event handler, all the event handlers execute simultaneously. They *do not* execute serially and they are not invoked in any particular order. (If you have a series of actions that must execute in a specific sequence, you should encapsulate the entire sequence within a single event handler.)
- An event handler can subscribe to more than one event.
- An event handler can be invoked synchronously or asynchronously. For more information, see "Invoking Event Handlers Synchronously or Asynchronously" on page 875.
- When event handlers run, they do not generate audit events.

- If an event handler throws an exception, it generates an exception event. This is true for all event handlers *but* exception event handlers. When an exception event handler throws an exception, it does not generate an exception event.

Subscribing to Events

You can use the Event Manager in Designer to subscribe to an event on the current server. This action registers the event handler with the Event Manager and specifies which events will invoke it.

Use the following procedure to subscribe to an event on the current Integration Server. Before you subscribe to an event, you must have completed the following:

- Identified the event type you want to subscribe to.
- Identified the service or services that generate an event you want to subscribe to (if you want to subscribe to an audit event, exception event, or JMS delivery failure event).
- Written the event handler that will execute when the identified event occurs.

To subscribe to an event

1. In Package Navigator view, select the current Integration Server and select **File > Properties**. In the Properties for `serverName` dialog box, select **Event Manager**.
2. In the **View event subscribers for** list, select the event type to which you want to subscribe.
3. Click to add a new subscriber.
4. In the Add Event Subscriber dialog box, complete the following fields:

In this field...	Specify...
Service	The fully qualified name of the event handler that will subscribe to the event (that is, the service that will execute when the event occurs). You can either type the name in the **Service** field or browse to locate and select the service from a list. **Example:** `sgxorders.Authorization:LogAuthTrans`
Filter	A pattern string to further limit the events this event handler subscribes to. Filters vary depending on the event type you are subscribing to. For example, if you are subscribing to an audit or exception event, create a filter to specify the names of services whose events this event handler subscribes to (that is, the services that, when executed, will invoke the event handler specified in **Service**).

In this field...	Specify...
	You can use the * character as a wildcard (this is the only wildcard character recognized by this pattern string). *The pattern string is case sensitive.* For more information about creating event filters, see "Creating Event Filters" on page 869.
Comment	An optional descriptive comment about this subscription.
Enabled	Whether the subscription is active or inactive. Set to **true** to activate the subscription. Set to **false** to deactivate the subscription. (This allows you to temporarily suspend a subscription without deleting it.)

5. Click **OK**. Subscriptions take effect immediately.

Note: Integration Server saves information for event types and event subscriptions in the eventcfg.bin file. This file is generated the first time you start the Integration Server and is located in the *Integration Server_directory* \config directory. Copy this file from one Integration Server to another to duplicate event subscriptions across servers.

Creating Event Filters

Event filters allow you to be very selective about the events to which you subscribe. Event filters limit the events for an event type that invoke an event handler. By using event filters, you can subscribe an event handler to only those events generated by a particular service, package, user, or port. For example, you might want an event handler to be invoked only when a specific service generates an audit event. Or, you might want an event handler to be invoked only when a specific user logs on to the Integration Server.

The following table identifies the information that you can filter on for each event type. Notice that you cannot create a filter for some event types. For these event types, every generated event invokes the event handlers subscribed to it.

Important:
The asterisk (*) is the only wildcard character allowed in an event filter. All other characters in the pattern string are treated as literals. Pattern strings are case sensitive.

For this event type...	You create a filter for...
Alarm Event	The message generated by the alarm event. Create a filter that specifies some of the text of the message. The event handler with this filter will process all alarm events containing the specified text. The following filter specifies that any alarm events that generate a message containing the word "port" will invoke the event handler: `*port*`

For this event type...	You create a filter for...
Audit Event	The fully qualified name of the service that generates the audit event. Create a filter to specify the services whose audit events you want to invoke the event handler. The following filter specifies that the service sgxorders.Authorization:creditAuth will invoke the event handler: `sgxorders.Authorization:creditAuth`
Audit Error Event	The concatenated value of the *destination* and *errorCode* fields of the audit error event. If the audit error event value matches the filter, the event will be passed to the event handler. You can use the asterisk (*) as a wildcard character in the filter. You can use filters to limit the events that your event handler will receive as follows: ■ If you set the filter to *YourSearchTerm*, the event handler will receive events whose values contain *only YourSearchTerm*. ■ If you set the filter to *YourSearchTerm**, the event handler will receive events whose values begin with *YourSearchTerm*. ■ If you set the filter to **YourSearchTerm*, the event handler will receive events whose values end with *YourSearchTerm*. ■ If you set the filter to **YourSearchTerm**, the event handler will receive events whose values contain *YourSearchTerm* anywhere in the value.
Error Event	The error message text. The following filter specifies that any error event with a message that contains the word "missing" will invoke the event handler. `*missing*`
Exception Event	The fully qualified name of the service that generates the exception event. Create a filter to specify the services whose exception events you want to invoke the event handler. The following filter specifies that all services that start with the word "credit" and belong to any folder will invoke the event handler: `*:credit*`
GD End Event	N/A The filter for all GD End events is the following: `*`

For this event type...	You create a filter for...
GD Start Event	The fully qualified name of the service that is being invoked using guaranteed delivery. Create a filter to specify the services that, when invoked using guaranteed delivery, will invoke the event handler. The following pattern string specifies that all services that start with the word "sendPO" and belong to any folder will invoke the event handler: `*:sendPO*`
JMS Delivery Failure Event	The name of the JMS connection alias used to send the message to the JMS provider. The following filter specifies that a JMS delivery failure event involving a JMS connection alias with "XA" in the JMS connection alias name will invoke the event handler: `*XA*`
JMS Retrieval Failure Event	The fully qualified name of the JMS trigger that called the trigger service for which the error occurred. The following filter specifies that a JMS retrieval failure event involving a JMS trigger named "ordering:processTransaction" will invoke the event handler: `*ordering:processTransaction*`
Journal Event	The major code and minor code of the generated event. The format of the filter is <majorCode>.<minorCode>. For example, the following filter specifies that any journal event with major code of 28 followed by a minor code of 34 will invoke the event handler: `*28.34*`
Port Status Event	N/A The filter for all port status events is the following: `*`
Replication Event	The name of the package being replicated. Create a filter to specify the packages that, when replicated, will invoke the event handler. The following filter specifies that a replication event involving the package named "AcmePartnerPkg" will invoke the event handler: `AcmePartnerPkg`
Security Event	N/A The filter for all security events is the following: `*`

For this event type...	You create a filter for...
Session End Event	N/A The filter for all session end events is the following: `*`
Session Expire Event	N/A The filter for all session expire events is the following: `*`
Session Start Event	The user name for the user starting the session on the Integration Server or the groups to which the user belongs. Create a filter to specify which users or which user groups invoke an event handler when they start a session on the server. The following filter specifies that a session start event generated by a user in the "Administrators" group will invoke the event handler. `*Administrators*`
Stat Event	N/A The filter for all stat events is the following: `*`
Tx End Event	N/A The filter for all Tx End events is the following: `*`
Tx Start Event	N/A The filter for all Tx Start events is the following: `*`

Creating Event Filters for Services

When you create a filter for a service name, you can be very selective about which service's events you subscribe to. You can use regular expressions to create event filters for service names. The following examples show ways you can use regular expressions as event filters to specify an event that a particular service generates. For more information about regular expressions, see "Regular Expressions" on page 1129.

This filter...	Will select events generated by...
`sgxorders.Auth:creditAuth`	The service sgxorders.Auth:creditAuth.

This filter...	Will select events generated by...
`sgxorders.Auth:credit*`	All services in the sgxorders.Auth folder, starting with the characters “credit.”
`sgxorders.Auth:*`	All services in the sgxorders.Auth folder.
`sgxorders.*`	All services in the sgxorders folder and its subfolders.
`*.Auth*:credit*`	All services starting with the characters “credit” that reside in any subfolder whose name starts the characters “Auth.”
`*:credit*`	All services starting with the characters “credit” in any folder.
`*`	All services.

Viewing and Editing Event Subscriptions

1. In Package Navigator view, select the current Integration Server and select **File > Properties**. In the Properties for *serverName* dialog box, select **Event Manager**.

2. In the **View event subscribers for** list, select the event type for which you want to view subscriptions.

3. Click the subscription you want to edit, and then click .

4. Modify the fields in the Edit Event Subscriber dialog box as needed and then click **OK**.

5. Repeat this procedure for each subscription that you want to view or edit.

6. Click **OK** when you finish viewing or editing event subscriptions. Your changes take effect immediately.

Suspending Event Subscriptions

You can suspend an event subscription. By suspending an event subscription, you temporarily stop the execution of the event handler without deleting or removing the event handler. While the event subscription is suspended, the Event Manager does not invoke the associated event handler when the server generates the event to which it is subscribed. You can resume an event subscription at any time.

To suspend an event subscription

1. In Package Navigator view, select the current Integration Server and select **File > Properties**. In the Properties for *serverName* dialog box, select **Event Manager**.

2. In the **View event subscribers for** list, select the event type for which you want to suspend a subscription.

3. Click the subscription you want to edit, and then click .

4. In the Edit Event Subscriber dialog box, in the **Enabled** list, select **false**.

5. Repeat this procedure for each event subscription you want to suspend.

6. Click **OK** when you finish suspending event subscriptions. Your changes take effect immediately.

Deleting an Event Subscription

1. In Package Navigator view, select the current Integration Server and select **File > Properties**. In the Properties for *serverName* dialog box, select **Event Manager**.

2. In the **View event subscribers for** list, select the event type for which you want to delete a subscription.

3. Click the subscription you want to delete, and then click .

4. Repeat this procedure for each subscription that you want to delete.

5. Click **OK** when you finish deleting subscriptions. Your changes take effect immediately.

Building an Event Handler

Building an event handler involves the following basic stages:

Stage 1	**Creating an empty service.** During this stage, you create the empty service that you want to use as an event handler.
Stage 2	**Declaring the input and output.** During this stage, you declare the input and output parameters for the event handler by selecting the specification or IS document type for the event type in pub.event. The specification and IS document type indicate the run-time data that will be contained in the IData object passed to the event handler.
Stage 3	**Inserting logic, code, or services.** During this stage, you insert the logic, code, or services to perform the action you want the event handler to take when the event occurs. If you are building a flow service, make sure to link data between services and the pipeline.

Stage 4	**Testing and debugging the service.** During this stage, you use the testing and debugging tools available in Designer to make sure the event handler works properly.
Stage 5	**Subscribing to the event.** During this stage, you use the Event Manager to subscribe the event handler to the event. This action registers the event handler with the Event Manager and specifies which events will invoke it. You can create filters to be more selective about the events to which you subscribe.

Invoking Event Handlers Synchronously or Asynchronously

By default, Integration Server invokes event handlers that subscribe to events synchronously. Once the event handler is invoked, Integration Server waits for a reply before executing the next step in the flow service. This configuration is useful for environments that do not allow the use of thread or for processes that require immediate responses.

You can configure Integration Server to process the event handlers asynchronously. In this case, once the event handler is invoked, Integration Server executes the next step in the flow service immediately. The server does not wait for a reply before continuing the execution of the service. Each process runs as a separate thread, thereby increasing the performance significantly.

There are server configuration parameters specific to each event type that you can use to specify whether the event handlers (services) that subscribe to the events are to be invoked synchronously or asynchronously. These server configuration parameters will be in the format: watt.server.event.*eventType*.async.

Set the value of the server configuration parameter specific to the event to true, if you want Integration Server to invoke the event handlers that subscribe to the event asynchronously. Set the value of the server configuration parameter specific to the event to false, if you want Integration Server to invoke the event handlers that subscribe to the event synchronously. The default value is true.

For more information about specifying the server configuration parameters, refer to *webMethods Integration Server Administrator's Guide*.

About Alarm Events

An alarm event occurs when Integration Server generates a message related to the status of the server. An alarm event can be generated for the following reasons:

- A client experiences a logon failure or is denied access to Integration Server. A client cannot log on because of "invalid credentials."
- Errors occur in the Cluster Manager. The inability to add a port to a cluster can cause errors in Cluster Manager.
- A user tries to access a port and is denied access to the port. (This can happen when a user tries to execute a service not allowed on the port.) This type of alarm event is sometimes called a *port access* exception.

- A port cannot be started. The most common reason a port cannot start is that the port is being accessed by another application.
- A service cannot be loaded or executed due to setup errors. For a flow service, a possible error is a missing XML metafile. For a Java service, possible errors include a missing class file or method.

You can use alarm events to invoke event handlers that execute when the server generates messages related to the status of the server. For example, you might want to create an event handler that notifies the administrator when a user is denied access to the server or to a port, when a service fails to load or execute, or when a port does not start. You can also create event handlers to send data to a network monitoring system.

About Audit Events

An audit event occurs when a service generates audit data. You can use the options in a service's **Audit** properties to specify when a service generates audit data. A service can generate audit data once, twice, or zero times during execution. You can use audit events to invoke other services when a particular service executes. For example, you might want an audit event generated for a critical service to invoke a logging service or a notification service. For more information about specifying when a service generates audit data, see "Configuring Service Auditing" on page 183.

About Audit Error Events

An audit error event occurs in the following situations:

- When a SQLException is encountered while trying to insert an audit record into the audit logging database.
- When Integration Server initializes and cannot connect to the audit logging database.
- When the Service logger is configured to retry failed auditing attempts, the audit error event is fired for the initial failure and each subsequent failure.

You can use audit error events to monitor your audit database for failures. For example, you could create a service that sends an email or a text message to the database administrator when the audit database becomes unavailable.

About Exception Events

An exception event occurs when a service throws an exception (including when a flow service "exits on failure"). You can use exception events to invoke some prescribed action, such as notifying an administrator, when a particular service fails.

Note:
Keep in mind that event handlers are processed independently of the services that invoke them. Event handlers are not designed to replace the error handling and/or error recovery procedures that you would normally include in your service.

If a nested service throws an exception, an exception event is generated by each service in the call stack. For example, if service A1 calls service B1, and B1 throws an exception, both B1 and A1 generate exception events (in that order).

About Guaranteed Delivery Events

A guaranteed delivery event occurs when a client uses guaranteed delivery to invoke a service on a remote Integration Server, and when the server returns the service results to the requesting client. There are two types of guaranteed delivery events:

- **GD Start events** occur when a client uses guaranteed delivery to invoke a service on a remote the Integration Server. In a flow service, executing the pub.remote.gd:start service generates a GD Start event.
- **GD End events** occur when a client receives the results of the service it requested using guaranteed delivery. In a flow service, executing the pub.remote.gd:end service generates a GD End event.

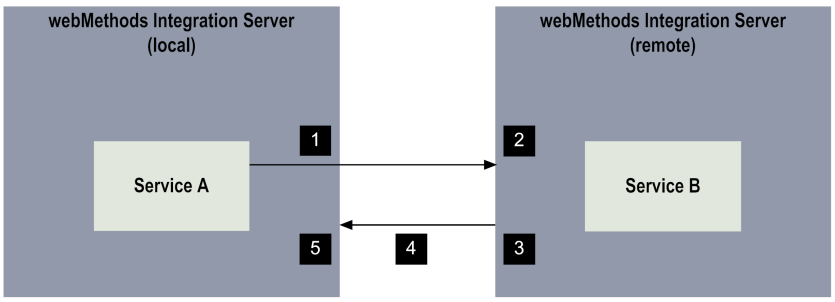
Each guaranteed delivery transaction generates a GD Start event and a GD End event. You can subscribe to GD Start and GD End events to invoke event handlers that log guaranteed delivery transactions to a file or database. You might also want to use guaranteed delivery events to invoke event handlers that send notification. For example, if you use guaranteed delivery to invoke a service that processes purchase orders, you might want to send notification to a business account manager about purchase orders from a particular client, or when the value of a purchase order is greater than a certain amount.

Guaranteed Delivery Events and Transaction Events

Guaranteed delivery events are related to transaction events (Tx Start and Tx End). Guaranteed delivery events begin when a client requests a guaranteed delivery transaction (GD Start) and when the client receives the results of the guaranteed delivery transaction (GD End). Transaction events occur when a service invoked using guaranteed delivery begins executing (Tx Start event) and when the service finishes executing (Tx End event).

The following diagram illustrates when guaranteed delivery events and transaction events occur during a guaranteed delivery transaction. In the following scenario, a local Integration Server uses guaranteed delivery to invoke a service on a remote server.

A Guaranteed Delivery Transaction generates Guaranteed Delivery Events and Transaction Events



Stage	Description
Stage 1	Service A uses guaranteed delivery to invoke Service B on the remote Integration Server. When the local server requests Service B, the local server generates a GD Start event. By default, the GD Start event is logged to the txout*yyyymmdd*.log file.
Stage 2	The remote Integration Server receives the request and begins executing Service B. When the remote server begins executing Service B, the remote server generates a Tx Start event. By default, the Tx Start event is logged to the txin*yyyymmdd*.log file.
Stage 3	The remote Integration Server finishes executing Service B and generates a Tx End event. By default, the Tx End event is logged to the txin*yyyymmdd*.log file.
Stage 4	The remote Integration Server sends the results of Service B to the requesting client (here, the local Integration Server).
Stage 5	The local Integration Server receives the results of Service B and generates a GD End event. By default, the GD End event is logged to the txout*yyyymmdd*.log file.

For details about guaranteed delivery, see the *Guaranteed Delivery Developer's Guide*.

About JMS Delivery Failure Events

Integration Server generates a JMS delivery failure event when a message written to the client side queue cannot be delivered to the JMS provider. When a transient error occurs, several delivery attempts may have been made.

You might want to create an event handler for a JMS delivery failure event to send notification or log information about the undelivered JMS message. You can also create an event handler that attempts to re-send the message to the JMS provider.

About JMS Retrieval Failure Events

By default, Integration Server generates JMS retrieval failure events when errors occur during message retrieval and JMS trigger processing. You can build event handlers that subscribe to and handle the JMS retrieval failure events.

A JMS retrieval failure event occurs in the following situations:

- A trigger service executed by a JMS trigger throws a non-transient error and the watt.server.jms.trigger.raiseEventOnException property is set to true (the default).
- A trigger service associated with a JMS trigger ends because of a transient error, all retry attempts have been made, and the JMS trigger is configured to throw an exception on retry failure. In addition, the watt.server.jms.trigger.raiseEventOnRetryFailure property is set to true (the default).

- The maximum delivery count from the JMS provider has been met for the message and the watt.server.jms.trigger.raiseEventOnRetryFailure property is set to true (the default).

  The watt.server.jms.trigger.maxDeliveryCount property specifies the maximum number of times the JMS provider can deliver a message to Integration Server. The default is 100. In a JMS message, the property *JMSXDeliveryCount* specifies the number of times the JMS provider delivered the message. Most JMS providers set this value.

- While performing exactly-once processing, the connection to the document history database is unavailable, and transient error handling for the JMS trigger is configured to **Throw exception** (non-transacted JMS trigger) or **Recover only** (transacted JMS trigger). In addition, the watt.server.jms.trigger.raiseEventOnRetryFailure property is set to true (the default).

- While performing exactly-once processing, the document resolver service ends with an ISRuntimeException, and transient error handling for the JMS trigger is configured to **Throw exception** (non-transacted JMS trigger) or **Recover only** (transacted JMS trigger). In addition, the watt.server.jms.trigger.raiseEventOnRetryFailure property is set to true (the default).

- While performing exactly-once processing, the document resolver service ends with an exception other than an ISRuntimeException. In addition, the watt.server.jms.trigger.raiseEventOnRetryFailure property is set to true (the default).

A service that functions as an event handler for a JMS retrieval failure event should use the pub.event:jmsReceiveErrorEvent specification as its service signature. For more information about the pub.event:jmsReceiveErrorEvent specification, see the *webMethods Integration Server Built-In Services Reference*.

About Port Status Events

A port status event occurs each time the Integration Server updates the server statistics. The port status event provides current status information about all of the configured ports on the Integration Server.

You can use port status events to invoke services that send port status data to a network monitoring system. You can also use port status events to invoke services that write port status data to a log file.

Note:
The watt.server.stats.pollTime property determines the frequency with which the Integration Server updates server statistics. The default frequency is 60 seconds. If you change this value, you must restart the Integration Server for the change to take effect. For more information about this property, see *webMethods Integration Server Administrator's Guide*.

About Replication Events

A replication event occurs when the pub.replicator:generateReplicationEvent executes. You might want to generate and subscribe to replication events to invoke event handlers that automate the completion of the package replication and distribution processes. For example, you could create replication event handlers that do the following:

- Notify package subscribers when a package is published.
- Maintain a log of replicated packages.
- Maintain a log of the packages distributed or "pushed" to your subscribers.
- Maintain a log of the packages your partners pulled from you.

For more information about the pub.replicator:generateReplicationEvent service, see the *webMethods Integration Server Built-In Services Reference*.

About Security Events

A security event occurs when an administrative or operational security action takes place on Integration Server and that security action is configured for auditing.

Administrative actions refer to configuration changes related to Integration Server security activities. Operational actions refer to successful and unsuccessful login attempts and access to Integration Server services, documents, and portlets.

Administrative security events include:

- Creating, modifying, and deleting packages, folders, and services.
- Creating, deleting, or modifying authentication related information. This includes creating new users, deleting users, changing their security attributes (for example, passwords), setting or modifying the mapping between certificates and users, and so on.
- Creating, deleting, or modifying authorization related information. This includes creating, modifying, and deleting ACLs.
- Creating, deleting, or modifying port settings. This includes defining allowed or denied actions on the port, port modes (allowed or denied by default in Integration Server), and certificate handling.
- Configuring SSL settings in Integration Server.
- Modifying or resetting Outbound Passwords.

Operational security events include:

- Successful logins to the Integration Server.
- Unsuccessful login attempts to the Integration Server. The login attempt failure could be because of incorrect password, disabled account, SSL failure, or expired certificate.
- Successful and unsuccessful accesses to services, files, and packages.
- Modifying existing passwords.
- Modifying messaging settings.

For information on configuring the Security logger, see *webMethods Audit Logging Guide*.

A service that functions as an event handler for a Security event should use the pub.event:security specification as its service signature. For more information about the pub.event:security service, see the *webMethods Integration Server Built-In Services Reference*.

About Session Events

A session event occurs when a client starts or ends a session on the Integration Server or when the Integration Server terminates an inactive session. You can subscribe to any of the following types of session events:

- **Session Start events** occur when a developer uses Designer to open a session on the Integration Server or when an IS client opens a session on the server to execute services.
- **Session End events** occur when a developer or IS client specifically issues a disconnect instruction to the Integration Server.
- **Session Expire events** occur when the Integration Server terminates an inactive session.

You can subscribe to session events to invoke event handlers that maintain your own log files or event handlers that send notification about users opening sessions on the server.

About Stat Events

A stat event occurs each time the Integration Server updates the statistics log (stats.log). The statistics log maintains statistical information about the consumption of system resources. The watt.server.stats.pollTime property determines the frequency with which the Integration Server updates statistics. The default frequency is 60 seconds.

You can use stat events to invoke event handlers that maintain your own log file or to invoke event handlers that send server statistics to a network monitoring system.

Note: Integration Server provides an agent that you can configure for use with a network monitoring system. For information about implementing this agent, see the readme file in the agentInstall.jar file located in the *Integration Server_directory* \lib directory.

About Transaction Events

A transaction event occurs when an Integration Server begins and finishes executing a guaranteed delivery transaction. There are two types of transaction events:

- **Tx Start events** occur when an Integration Server begins executing a service invoked with guaranteed delivery.
- **Tx End events** occur when an Integration Server finishes executing a service invoked with guaranteed delivery.

Transaction events result from guaranteed delivery transactions. Each guaranteed delivery transaction generates a Tx Start event and a Tx End event. In fact, the transaction events occur between the guaranteed delivery events. A Tx Start event occurs immediately after a GD Start event and a Tx End event occurs immediately before a GD End event. For more information about

how transaction events relate to guaranteed delivery events, see "Guaranteed Delivery Events and Transaction Events" on page 877.

You can subscribe to Tx Start and Tx End events to invoke event handlers that log guaranteed delivery transactions to a file or database. You might also want to use transaction events to invoke event handlers that send notification.

41 Submitting and Receiving XML Documents

- Submitting and Receiving XML in a String Variable 885
- Submitting and Receiving XML in $xmldata 886
- Submitting and Receiving XML via HTTP 888
- Submitting and Receiving XML via FTP 893
- Submitting and Receiving XML via E-mail 896

You can create a client that submits an XML document to a target service, which then receives the XML document.

The following table describes the methods a client can use to submit an XML document and how Integration Server passes the XML document to the target service based on the method.

Method	Integration Server Action
Submit the XML document in an arbitrarily named String variable	Integration Server passes the document as an XML String to the target service. It is the responsibility of the target service to parse the XML so that it is in a format that can be manipulated. For more information, see "Submitting and Receiving XML in a String Variable" on page 885.
Submit the XML document in a special String variable named *$xmldata*	Integration Server automatically parses the XML and passes it as a node to the target service. For more information, see "Submitting and Receiving XML in a String Variable" on page 885. **Note:** For information about submitting the XML document in the *$xmldata* variable, but bypass the automatic parsing, see "Submitting and Receiving XML via $xmldata without Parsing" on page 892.
Post the XML document via HTTP	Integration Server either automatically parses the XML and passes it as a node to the target service or passes the XML document directly to the target service as an XML stream or byte array. For more information, see "Submitting and Receiving XML in a String Variable" on page 885.
FTP the XML document	Integration Server automatically parses the XML and passes it as a node to the target service. For more information, see "Submitting and Receiving XML in a String Variable" on page 885.
Send the XML document as an email attachment	Integration Server automatically parses the XML and passes it as a node to the target service. For more information, see "Submitting and Receiving XML in a String Variable" on page 885.

Submitting and Receiving XML in a String Variable

A client can submit an XML document to Integration Server by passing the entire XML document to a target service in an arbitrarily named String variable.

Sample Client Code to Submit an XML Document in a String Variable

The following sample code fragment for a Java client illustrates how to submit an XML document in a String variable named *orders* to the target service, purch:postOrder, on Integration Server. This code fragment performs the following steps:

1. Loads the XML document into a String.
2. Puts the String into a variable named *orders* in an IData object named *inputs*.
3. Invokes purch:postOrder on the server at localhost:5555.

```
import com.wm.app.b2b.client.*;
import com.wm.util.*;
import com.wm.data.*;
import java.io.*;
public class ArbitraryXMLClient
     .
     .
     .
```

```
    //--Load XML into a String named orders
```

```
        String orders = YourLoadXMLMethod(orderFile);
```

```
    //--Put input values into the IData object
        IData inputs = IDataFactory.create();
        IDataCursor inputsCursor = inputs.getCursor();
```

```
       inputsCursor.insertAfter("orders", orders);
```

```
        inputsCursor.insertAfter("authcode", authcode);
```

```
    //--Submit request to the server at localhost:5555
         c.connect("localhost:5555", "null", null);
```

```
       IData outputs = c.invoke("purch", "postOrder", inputs);
```

```
         c.disconnect();
    .
    .
    .
```

Considerations When Coding the Target Service to Receive the XML Document that is Passed in a String Variable

When a client submits an XML document to a target service in an arbitrarily named String variable, the target service must take action to parse the XML so that the service can then subsequently manipulate the data in the XML document.

Code the target service to invoke the pub.xml:xmlStringToXMLNode service, passing the xmlStringToXMLNode service the String variable that contains the XML document. The xmlStringToXMLNode service produces a node that the target service can subsequently query or convert to an IData object.

For example, continuing with the previous example, the target service, purch:postOrder, would pass the *orders* String, which contains the XML document, to pub.xml:xmlStringToXMLNode.

After the XML document is represented as a node, the target service can invoke:

- pub.xml:queryXMLNode to query the node
- pub.xml:xmlNodeToDocument to convert the node to an IData object

For more information about the pub.xml:xmlStringToXMLNode. pub.xml:queryXMLNode, and pub.xml:xmlNodeToDocument services, see *webMethods Integration Server Built-In Services Reference*.

Submitting and Receiving XML in $xmldata

A client can submit an XML document to Integration Server by passing the entire XML document to a target service in a String variable named *$xmldata*.

Submitting an XML document to Integration Server using the *$xmldata* variable is similar to submitting it in an arbitrarily named String variable. However, the *$xmldata* variable has special meaning to Integration Server. When Integration Server receives the *$xmldata* variable, the server assumes the variable contains an XML document and automatically parses the XML and passes it to the target service as a node.

Note:
To use the *$xmldata* variable to submit an XML document, but bypass automatic parsing so that Integration Server sends the body of the request directly to the target service as a stream or byte array, your client must use HTTP to invoke the target service. For more information, see "Submitting and Receiving XML via $xmldata without Parsing" on page 892.

Sample Client Code to Submit an XML Document in $xmldata

The following sample code for a Java client that runs in Integration Server illustrates how to read an XML document from a file, assign the XML to a variable named *$xmldata*, and invoke the target service sales:getOrder passing it the *$xmldata* variable.

```
import com.wm.app.b2b.client.*;
import com.wm.util.*;
import com.wm.data.*;
import java.io.*;
```

```
public class ArbitraryXMLClient

{
   public static void main(String args[])
       throws Exception
   {
       //--Read the XML document from a specified file (or from stdin)
           Context c = new Context();
           IData inputs = IDataFactory.create();
           IDataCursor inputsCursor = inputs.getCursor();
           Reader in = null;
           if (args.length > 0)
           {
              in = new InputStreamReader(new FileInputStream(args[0]));
           }
           else
              in = new InputStreamReader(System.in);
           }
           char[] buf = new char[8192];
           int count = 0;
           StringBuffer sb = new StringBuffer();
           while((count = in.read(buf)) != -1)
           {
              sb.append(buf, 0, count);
           }

       //--Assign the XML document to a String variable

           String xmldata = sb.toString();


       //--Put the XML document into $xmldata in the IData object

           inputsCursor.insertAfter("$xmldata", xmldata);


       //--Submit the request to the sales:getOrder service on the server
            c.connect("localhost:5555", "null", null);

           IData outputs = c.invoke("sales", "getOrder", inputs);

           c.disconnect();

       //--Display the returned output values
           System.out.println(outputs);
   }

}
```

Important:
This example shows a Java-based client. However, you can use any type of IS client, including a browser-based client. For a browser-based client, post the XML document as the value portion of a `$xmldata=`*value* pair. You can post other name=value pairs with the request. For more information, see "Building a Browser-Based Client" on page 913.

Considerations When Coding the Target Service to Receive the XML Document that is Passed in $xmldata

BecauseIntegration Server passes the parsed XML document in the *node* variable, the target service that receives the XML document must take a *node* variable as input.

The target service can then manipulate the data in the XML node. The target service can pass the node to another service that takes *node* as input, for example:

- pub.xml:queryXMLNode to query the node
- pub.xml:xmlNodeToDocument to convert the node to an IData object

For more information about the pub.xml:queryXMLNode and pub.xml:xmlNodeToDocument services, see *webMethods Integration Server Built-In Services Reference*.

Submitting and Receiving XML via HTTP

A client can post an XML document to a target service that receives the XML document via HTTP.

When Integration Server receives an HTTP request and the Content-Type field in the request header is `text/xml` or `application/xml`, Integration Server performs one of the following actions:

- Automatically parses the XML and passes it as a node to the target service.
- Passes the XML document directly to the target service as an XML stream or byte array.

The action that Integration Server takes depends on the xmlFormat value. By default, Integration Server obtains the xmlFormat value from the **Default xmlFormat** property for the target service. If no value is specified for the property, Integration Server obtains the value from the watt.server.http.xmlFormat server configuration parameter. However, the client can override the assigned property by supplying the xmlFormat argument in the URL it uses to invoke the target service. For more information about supplying the xmlFormat value in the request URL, see "About the xmlFormat Value" on page 890.

Creating a Client that Submits an XML Document via HTTP

To submit an XML document to Integration Server via HTTP, create a client that sends the XML document as a byte array to Integration Server using the HTTP POST or HTTP PUT methods. When constructing the HTTP request, the client needs to:

- Address the request to the URL of the target service. For example, if the purch:postOrder service is to receive the XML document, the client might use the following URL:

  http://rubicon:5555/invoke/purch/postOrder

- Set the Content-Type header field in the HTTP request header to `text/xml`.

  Because most browsers do not allow you to modify the Content-Type header field, they are not suitable clients for this type of submission. Clients that you might use to submit an XML

document in this manner are PERL scripts, which allow you to build and issue HTTP requests, or the Integration Server pub.client:http service.

- Place the XML document in the body of the HTTP request. Place an extra carriage return/new line (\r\n) at the end of it to indicate the end of the XML document.

Important:
The XML document must be the only text in the body of the request. Do not assign the XML document to a name=value pair.

Using pub.client:http to Submit an XML Document via HTTP

If the client is a service running in an Integration Server, the client can invoke the pub.client:http service to submit the XML document via HTTP.

The following table describes how to set the input variables for the pub.client:http service to POST an XML document to a target service. For a full description of the pub.client:http service, see *webMethods Integration Server Built-In Services Reference*

pub.client:http input variable	Description
url	Specify the URL of the target service that is to receive the XML document. In the URL, include the xmlFormat argument if you want to override the behavior specified by the **Default xmlFormat** property for the target service. For more information about the xmlFormat values, see "About the xmlFormat Value" on page 890. ■ If you want Integration Server to take action on the XML document based on the specified **Default xmlFormat** property value for the target service, use the following format for the URL: http://*hostname*:*port*/invoke/*folder*/*serviceName* For example, the following URL invokes the purch:postOrder service on the server at rubicon:5555. http://rubicon:5555/invoke/purch/postOrder ■ If you want to override the **Default xmlFormat** property value specified for the target service, use the following format for the URL: http://*hostname*:*port*/invoke/*folder*/*serviceName*?xmlFormat=*format* Where format can be `enhanced`, `node`, `stream`, or `bytes`. For more information about the xmlFormat value and how it impacts processing of the XML document, see "About the xmlFormat Value" on page 890. **Note:**

pub.client:http input variable	Description
	The client request should specify the xmlFormat argument only when it is recommended in the documentation for the target service. Furthermore, a client should specify the xmlFormat only when knowing how the service will respond.
method	Specify `post` or `put`.
headers	Specify information for the Content-Type field of the HTTP request header.

Key	Value
Content-Type	`text/xml` or `application/xml`

data Specify the XML document to submit via HTTP. Use one of the following keys:

Key	Value
string	A string containing the XML document to submit.
bytes	A byte array containing the XML document to submit.

About the xmlFormat Value

When a client submits an XML document via HTTP, the action that Integration Server takes when it receives the XML document is based on the xmlFormat value. The xmlFormat value determines:

- Whether Integration Server parses the XML document automatically before passing it to the service
- Which XML parser Integration Server uses. When parsing an XML document, Integration Server uses either the legacy XML parser or the enhanced XML parser. For more information about the XML parsers, see *webMethods Integration Server Administrator's Guide*.
- The name and data type of the variable that Integration Server adds to the pipeline with the contents of the XML document

The following table describes the actions that Integration Server can take based on the xmlFormat value:

xmlFormat value	Integration Server Action
bytes	Integration Server passes the XML document directly to the service as a byte array without parsing the XML. Integration Server places the byte array in the input pipeline of the target service in a variable named *xmlBytes*.

xmlFormat value	Integration Server Action
enhanced	Integration Server parses the XML using the enhanced XML parser automatically. Integration Server uses the default options specified for enhanced XML parsing on the **Settings > Enhanced XML Parsing** page in Integration Server Administrator. Integration Server passes the XML document to the target service as a org.w3c.dom.Node object named *node*. For more information about configuring the enhanced XML parser, see *webMethods Integration Server Administrator's Guide*.
node	Integration Server parses the XML using the legacy XML parser automatically and passes it to the target service as a com.wm.lang.xml.Node named object *node*.
stream	Integration Server passes the XML document directly to the service as an XML stream without parsing the XML. Integration Server places the XML stream in the input pipeline of the target service as an InputStream named *xmlStream*.

Note:
If parsing is not needed, it can unnecessarily slow down the execution of a service. For example, an application might handle the XML as a simple String. In this case, the automatic parsing is unnecessary and should be avoided.

By default, Integration Server obtains the xmlFormat value from the **Default xmlFormat** property assigned to the target service. However, the client can override the **Default xmlFormat** property value by supplying the xmlFormat argument in the URL it uses to invoke the target service. The following shows the URL format when using the xmlFormat argument:

http://*hostname*:*port*/invoke/*folder*/*serviceName*?xmlFormat=*format*

Specify `cached`, `node`, `stream`, or `bytes` for *format*.

For example, suppose that the configured **Default xmlFormat** property value is `node`. If you want to invoke the sales:orderInfo service on the server rubicon:5555 and override the configured **Default xmlFormat** value so that Integration Server passes the XML document directly to the sales:orderInfo service as an XML stream, use the following URL:

http://rubicon:5555/invoke/sales/orderInfo?xmlFormat=stream

Note:
The client request should specify the xmlFormat argument only when it is recommended in the documentation for the service. Furthermore, a client should specify the xmlFormat argument only when knowing how the service will respond.

Submitting and Receiving XML via $xmldata without Parsing

When submitting an XML document in $xmldata, you might not want Integration Server to automatically parse the XML and pass it to the target service as a node, as described in "Submitting and Receiving XML in $xmldata" on page 886. If parsing is not needed, it can unnecessarily slow down the execution of a service. For example, an application might handle the XML as a simple String. In this case, the automatic parsing is unnecessary and should be avoided.

Using pub.client:http to Submit $xmldata via HTTP

If the client is a service running in an Integration Server, the client can invoke the pub.client:http service to submit an XML document in $xmldata via HTTP.

The following table describes how to set the input variables for the pub.client:http service use $xmldata to submit an XML document and to include the xmlFormat argument in the URL to override the watt.server.http.xmlFormat setting. For a full description of the pub.client:http service, see *webMethods Integration Server Built-In Services Reference*.

pub.client:http input variable	Description
url	Specify the URL of the target service that is to receive the XML document. **Note:** Rather than specifying the query string portion of the URL with the xmlFormat argument in the *url* variable, specify the xmlFormat argument in the *data/args* variable, as described below.
method	Specify `post`.
data	Use the *args* key of the *data* input variable to specify key/value pairs that the service places in the query string of the URL.

Key	Value
args	A Document containing the arguments you want the service to add to the URL in the query string. Specify the following arguments:

Argument	Value
$xmldata	A String that contains the XML document that the client wants to submit via HTTP. **Note:** Using $xmldata for the XML document indicates the data is XML. As a result, you do not need to use the *headers* input variable to specify the Content-Type field of the HTTP request header.

pub.client:http input variable	Description
xmlFormat	A String that indicates how you want Integration Server to pass the XML document to the target service. Specify `stream` or `bytes`. ■ Use `stream` if you want Integration Server to pass the XML document as an XML stream without parsing the XML. When passing the XML document as an XML stream, Integration Server places the XML in the input pipeline of the target service as an InputStream named *xmlStream*. ■ Use `bytes` if you want Integration Server to pass the XML document as a byte array without parsing the XML. When passing the XML document as a byte array, Integration Server places the XML in the input pipeline of the target service as a byte array named *xmlBytes*. The query string that the service appends to the URL will use the following format: ?$xmldata=*string*&xmlFormat=*format* where: ■ *string* is the value you specify for the $xmldata argument (that is, the XML document). ■ *format* is the value you specify for the xmlFormat argument.

For information about how to set other input variables when using the pub.client:http service to submit an XML document, see "Submitting and Receiving XML via HTTP" on page 888.

Submitting and Receiving XML via FTP

A client can FTP an XML document to the Integration Server FTP listening port.

To FTP the XML document, the client sends the XML document in a file to the target directory. The target directory is the Integration Server namespace (ns) directory that contains the target service that is to receive the XML document. When the Integration Server receives the file on the FTP listening port, the server automatically parses the XML document and passes it as a node to the target service.

Naming the File that the Client is to Submit via FTP

When specifying the file name of the file that contains the XML document, use a file extension that Integration Server recognizes as containing XML content. The FTP listener chooses a content handler based on the file extension. You can use one of the following for the file extension:

- A file extension of "xml".
- An arbitrary file extension that you have registered with the `text/xml` content type.

  To register a file extension with a content type, edit the Integration Server lib\mime.types file, which contains the mappings of file extensions to content types. You can edit the mappings in the lib/mime.types file from Integration Server Administrator by selecting **Settings > Resources > Mime Types**.

  For example, if you want to submit an XML document in a file that has the file extension xml2, add the following to the lib/mime.types file to register xml2 and assign it the `text/xml` content type:

  ```
  text/xml     xml2
  ```

- No file extension

  If you want to submit an XML document in a file that has no file extension, edit the lib/mime.types file and add the following to associate the special key, `ftp_no_extension` with the `text/xml` content type. Using `ftp_no_extension` indicate a null extension.

  ```
  text/xml ftp_no_extension
  ```

Actions a Client Takes to Submit an XML Document via FTP

To submit an XML document to a target service via FTP, code the client to:

1. Initiate an FTP session on theIntegration Server FTP listening port.

   By default the FTP port is assigned to port 8021. However, this assignment is configurable, so check with the Integration Server administrator to determine the port number to use for FTP communications with Integration Server.

2. Point to the target directory where the client is to copy the file containing the XML document.

   The target directory is the Integration Server namespace (ns) directory where the target service resides. Use the following format:

   ```
   cd \ns\folder\subfolder\serviceName
   ```

   For example, if the target directory is the namespace directory containing the purchasing:submitOrder service, use the following:

   ```
   cd \ns\purchasing\submitOrder
   ```

   Important:

Note that the root directory for this operation is your Integration Server's namespace directory (ns), not the root directory of the target machine.

3. Copy the XML document to the target directory using the following command, where *filename* is the name of the file that contains the XML document:

```
put filename
```

For example, to copy the PurchaseOrder.xml2 file, use the following:

```
put PurchaseOrder.xml2
```

The file that the client sends to Integration Server via FTP is never actually written to the server's file system. The XML document you send and the output file it produces are written to a virtual directory system maintained in the client's Integration Server session. When the client ends the FTP session, Integration Server automatically deletes the original file and any results from the session.

Important: Software AG recommends that you use a unique name for each XML document that you FTP to Integration Server (perhaps by attaching a timestamp to the name) so that you do not inadvertently overwrite other FTPed XML documents or their results during an FTP session.

If your client is a service running in an Integration Server, instead of coding each of the actions described above, the client can invoke services in the pub.client folder to FTP a file. For information about these services, see the *webMethods Integration Server Built-In Services Reference*.

Actions a Client Takes to Retrieve Output from the Target Service

If the target service returns output, Integration Server writes the results to the same virtual directory where the XML document was initially sent via FTP. The results are in an output file with a name that uses the following format where *filename* is the name of the XML file originally sent to the target service via FTP:

filename.out

For example, if the name of the XML file was PurchaseOrder.xml2, Integration Server writes the results to the following file:

PurchaseOrder.xml2.out

Code the client to retrieve the output file using the FTP "get" command. For example, to retrieve the output in PurchaseOrder.xml2.out, the client can use the following FTP command:

```
get PurchaseOrder.xml.out
```

If your client is a service running in an Integration Server, it can invoke services in the pub.client folder to perform FTP commands to get a file. For information about these services, see the *webMethods Integration Server Built-In Services Reference*.

Considerations When Coding the Target Service to Receive the XML Document

Because Integration Server passes the parsed XML document in the *node* variable, the target service that receives the XML document must take a *node* variable as input.

The target service can then manipulate the data in the XML node. The target service can pass the node to another service that takes *node* as input, for example:

- pub.xml:queryXMLNode to query the node
- pub.xml:xmlNodeToDocument to convert the node to an IData object

For more information about the pub.xml:queryXMLNode and pub.xml:xmlNodeToDocument services, see *webMethods Integration Server Built-In Services Reference*.

If the target service returns output, the format of the returned output depends on whether you assign an output template to the target service:

- If the service does not have an output template assigned to it, the results from the service (that is, the contents of the pipeline) are XML-encoded and returned as an XML document.
- If the service has an XML output template assigned to it, that template is applied to the results. (If the template is not an XML-based template, it is not applied.)

Integration Server writes the output to the same virtual directory where the XML document was initially sent via FTP. The results are in a file named *filename*.out, where *filename* is the name of the XML file originally sent to the target service via FTP.

Submitting and Receiving XML via E-mail

A client can email an XML document to an email mailbox that Integration Server monitors. When the email message arrives, Integration Server automatically retrieves the message and processes the XML document contained in the message.

To use this method to submit an XML document, Integration Server must be configured with an email port that monitors the mailbox to which the XML document will be submitted. Consult your Integration Server administrator to determine whether Integration Server has a defined email port.

When Integration Server receives an XML document via email, the server automatically parses the XML document and passes it as a node to a service for processing.

Actions a Client Must Take to Submit an XML Document via Email

To submit an XML document to Integration Server via email, the client must:

- Place the XML document in an email attachment.
- Set the attachment's Content-Type header to `text/xml` or `application/xml`.

- Identify the name of the target service that is to process the XML document in the subject line of the email message.

  Note:
  If you leave the subject line empty, Integration Server first attempts to pass the XML document to the global service. If the global service is not defined, the server then attempts to pass the XML document to the default service assigned to the email port (if one has been assigned). You assign the global service and the port's default service when defining the email port. For more information, see *webMethods Integration Server Administrator's Guide*.

Using pub.client:smtp to Submit an XML Document via Email

If the client is a service running in an Integration Server, the client can invoke pub.client:smtp to submit an XML document via email.

The following describes the input values to supply when invoking pub.client:smtp to submit an XML document as an attachment of an email message. For more information about using this service, see the *webMethods Integration Server Built-In Services Reference*.

pub.client:smtp input variable	**Description**
to	A String containing the email address that the Integration Server email port monitors.
subject	A String containing the fully qualified name of the target service that is to process the XML document. For example: `orders:ProcessPO` **Note:** If *subject* is not specified, Integration Server passes the XML document to the global service if it is defined. If the global service is not defined, Integration Server passes the XML document to the default service assigned to the email port, if one is assigned.
from	A String containing the email address where the client expects results. The target service should send its output to this email address.
body	A String containing input variables for the target service in URL query string format. For example: `one=1&two=2&three=3&$user=Administrator&$pass=manage` This example sets five input variables: *one*, *two*, and *three* are set to the values 1, 2 and 3, respectively. The input variables *$user* and *$pass* have special meaning to the email port. Use these variables to specify the user name and password for the email port. You must specify *$user* and *$pass* if authentication is enabled on the email port.

pub.client:smtp input variable	Description
attachments	A document list containing a single document that specifies the XML document to submit via email.

Key	Value
contenttype	A String containing the content-type of the attachment. Set *contenttype* to `text/xml`.
content or *filename*	The XML document. ■ Specify *content* to provide a byte array containing the XML document. ■ Specify *filename* to specify the fully qualified name of the file containing the XML document.

Considerations When Coding the Target Service to Receive the XML Document

BecauseIntegration Server passes the parsed XML document in the *node* variable, the target service that receives the XML document must take a *node* variable as input.

The target service can then manipulate the data in the XML node. The target service can pass the node to another service that takes *node* as input, for example:

- pub.xml:queryXMLNode to query the node
- pub.xml:xmlNodeToDocument to convert the node to an IData object

For more information about the pub.xml:queryXMLNode and pub.xml:xmlNodeToDocument services, see *webMethods Integration Server Built-In Services Reference*.

If the you want the target service to return output to the client:

- Configure the email port to return results from requests that it receives.

  By default, the email port does not return results. For information about configuring the email port to return results, see *webMethods Integration Server Administrator's Guide*.

  If your email port is configured to return results, Integration Server emails the output from target service back to the sender of the original message, in an attachment file called xml.out.

- Define how you want the target service to return its output.
  - If the service does not have an output template assigned to it, the results from the service (that is, the contents of the pipeline) are XML-encoded and returned as an XML document.
  - If the service has an XML output template assigned to it, that template is applied to the results. (If the template is not an XML-based template, it is not applied.)

42 Working with Load and Query Services

- What Are the Load and Query Services? 900
- Basic Concepts 900
- About the pub.xml:loadXMLNode Service 901
- About the pub.xml:loadEnhancedXMLNode Service 901
- About the pub.xml:queryXMLNode Service 901

You can use the *load and query* services to fetch HTML or XML documents from the Internet and extract data for use in other services.

What Are the Load and Query Services?

Integration Server is equipped with a set of "load and query" services that allow you to fetch HTML or XML documents via HTTP or HTTPS and selectively extract information for use in other services. Using these services, you can connect a service to virtually any document on the Internet. Fetching an XML document from the Internet with Integration Server is a two-step process.

- First, you use the pub.xml:loadXMLNode service or the pub.xml:loadEnhancedXMLNode service to retrieve the document that contains the information you need.
- Next, you extract the pieces of information you need and assign them to Integration Server variables. For this step, you can use the pub.xml:queryXMLNode service (to select specific elements from the document) or the pub.xml:xmlNodeToDocument service (to convert all elements in the document to variables).

Note:
If you want to retrieve documents from a local file system, use the pub.file:getFile service. For more information about pub.file:getFile, see the *webMethods Integration Server Built-In Services Reference*.

Basic Concepts

To successfully use Integration Server's load and query services, you should understand the following terms and concepts.

Term	Concept
Parsing	Parsing is the operation that the server performs to convert an XML document (a string) into an XML node whose elements can be addressed and extracted by services. Integration Server automatically parses XML documents that you fetch with the pub.xml:loadXMLNode service.
Node	A node is the result of a parsing operation. It is an element-based representation of an XML document. The node expresses a document in a tree-like structure that allows the data within it to be efficiently addressed and linked into services.
Query	A query is an expression, written in either the XML Query Language (XQL) or the webMethods Query Language (WQL) that you use to extract (filter) information from XML nodes.
WQL	WQL (webMethods Query Language) is a language that is used to retrieve information from an HTML or XML document. For more information about WQL, see " webMethods Query Language" on page 1151.
XQL	XQL (XML Query Language) is a language that is used to retrieve information from an XML document.

About the pub.xml:loadXMLNode Service

You use the pub.xml:loadXMLNode service to retrieve and parse an XML or HTML document from the Internet. This service does the following:

- First, it submits a HTTP or HTTPS request for a specified XML or HTML document.
- Then, it parses the returned document using the legacy XML parser.

The output from pub.xml:loadXMLNode is an XML node that can be used with any service that takes a *document* or a *node* as input.

Note:
If you want to fetch a document from a local file system, do not use pub.xml:loadXMLNode. Instead, use the pub.file:getFile service. For more information, see the *webMethods Integration Server Built-In Services Reference*.

About the pub.xml:loadEnhancedXMLNode Service

You use the pub.xml:loadEnhancedXMLNode service to retrieve and parse an XML document from the Internet. This service does the following:

- First, the service submits a HTTP or HTTPS request for a specified XML document.
- Then, the service parses the returned document using the enhanced XML parser.

The output from pub.xml:loadEnhancedXMLNode is an org.w3c.dom.Node object. A DOM node is a special representation of an XML document that can be consumed by any program that uses standard DOM APIs. The pub.xml:xmlNodeToDocument service can accept a DOM object as input.

Note:
If you want to fetch a document from a local file system, do not use pub.xml:loadEnhancedXMLNode. Instead, use the pub.file:getFile service. For more information, see the *webMethods Integration Server Built-In Services Reference*.

About the pub.xml:queryXMLNode Service

You use the pub.xml:queryXMLNode service to selectively extract information from a parsed HTML or XML document and assign that information to variables that can be linked to other services.

When you use pub.xml:queryXMLNode, you extract information using either the XML Query Language (XQL) or the webMethods Query Language (WQL). Both languages allow you to address and select information from a node based on criteria that you specify in a query statement. For more information about WQL, see " webMethods Query Language" on page 1151 .

Note:
When you use pub.xml:queryXMLNode to query an enhanced XML node (a node produced by the enhanced XML parser), you must use XQL as the query language.

43 Building Services that Retry

- Requirements for Retrying a Service 904
- Example Service that Throws an Exception for Retry 905

When creating a service, you can construct and configure the service to retry automatically if a transient error occurs during service execution. A *transient error* is an error that arises from a temporary condition that might be resolved or restored, such as the unavailability of a resource due to network issues or failure to connect to a database. The service might execute successfully if Integration Server waits a short interval of time and then retries the service.

To build a service that retries, you create it so that it catches errors and determines whether an error is transient. When the service determines that an error is transient, have it re-throw the error as an ISRuntimeException. The ISRuntimeException is the signal to Integration Server to retry the service. For more information about how to construct the service, see "Requirements for Retrying a Service" on page 904 and "Example Service that Throws an Exception for Retry" on page 905.

In addition to constructing the service for retry, you also must set retry properties for the service (or the trigger calling the service) so that Integration Server knows that it is to retry a service when an ISRuntimeException is thrown. When Integration Server retries the service it re-executes it using the original input. For more information about how to configure a service for retry, see "About Automatic Service Retry" on page 178 and "Configuring Service Retry" on page 179.

Requirements for Retrying a Service

If you want a service to catch a transient error, re-throw it as an ISRuntimeException, and then re-execute, the following criteria must be met:

- You must configure the **Transient error handling** properties for the top-level service. For more information about configuring service retry and how Integration Server retries services, see "Configuring Service Retry" on page 179.
- If the service functions as a trigger service (the service is invoked by a trigger), you must configure the **Transient error handling** properties for the trigger. In this situation, Integration Server uses the trigger retry properties instead of the service retry properties. For more information about configuring retry for webMethods Messaging Triggers, see the "Configuring Transient Error Handling for a webMethods Messaging Trigger " on page 720. For information about configuring retry for JMS triggers, see the "Configuring Transient Error Handling for a Non-Transacted JMS Trigger" on page 662.
- If the service is a flow service, the service must invoke pub.flow:throwExceptionForRetry to throw the ISRuntimeException. For more information about the pub.flow:throwExceptionForRetry service, see the *webMethods Integration Server Built-In Services Reference*.
- If the service is written in Java, the service can use com.wm.app.b2b.server.ISRuntimeException() to throw the ISRuntimeException. For more information about constructing ISRuntimeExceptions in Java services, see *webMethods Integration Server Java API Reference* for the com.wm.app.b2b.server.ISRuntimeException class.
- If the service invokes an adapter service, ensure that the service catches transient errors that the adapter service detects.

  When an adapter service built on Integration Server 6.0 or later, and based on the ART framework, detects a transient error, for example, if their back-end server is down or the network connection is broken, the adapter service propagates an exception that is based on ISRuntimeException. When creating a service that invokes an adapter service, ensure that the

logic that catches errors and determines whether they are transient errors can interpret the adapter service exception that signals a retry.

For more information about adapter services, see the relevant adapter guides.

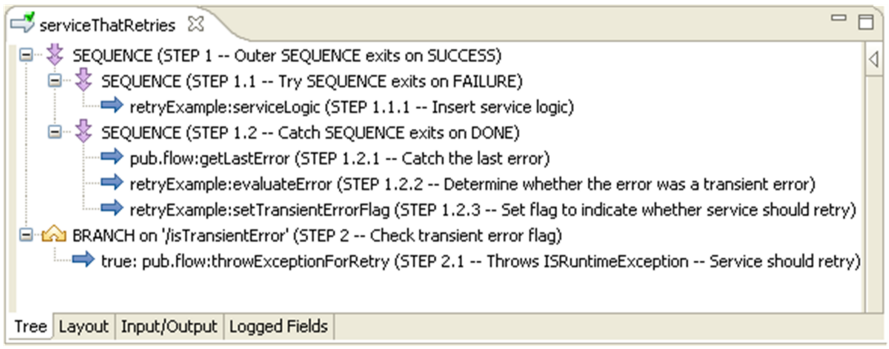
Example Service that Throws an Exception for Retry

This example shows one possible way to build a service that catches errors, checks errors to determine whether they are transient, and re-throws an error as an ISRuntimeException if it is transient. It includes the following basic sections of logic:

- **An outer sequence** that contains a try sequence and a catch sequence:
  - **The try sequence** executes the work that you want the service to perform.
  - **The catch sequence** examines any exception that occurs in the try sequence, determines whether the exception is a transient error, and indicates whether Integration Server should retry the service.

  The outer sequence is used so that the catch sequence is skipped when the try sequence is successful.
- **A throw exception step** that executes only when the catch sequence indicates that a transient error occurred. It throws the ISRuntimeException to signal Integration Server that the service should be re-executed.

Example logic for a service that throws an exception for retry



- **STEP 1 - Outer SEQUENCE exits on SUCCESS**

  The outer sequence wraps the try sequence and the catch sequence. This sequence is set to exit on success so that it exits when either of its child steps (the try sequence or the catch sequence) execute successfully. If the try sequence executes successfully, Integration Server skips the catch sequence.

- **STEP 1.1 - Try SEQUENCE exits on FAILURE**

  The try sequence contains the logic you want the service to execute. This sequence is set to exit on failure. As a result, if a step in this try sequence fails, Integration Server executes the next step, which is the catch sequence that checks for transient errors.

- **STEP 1.1.1 - Insert service logic**

  This step contains the logic that you want the service to perform. The service logic might consist of multiple services or flow steps.

  When the service logic executes successfully, that means the try sequence is successful. Because the outer sequence exits on success, Integration Server exits the outer sequence, skipping the catch sequence. Integration Server executes the next step, which is the BRANCH on '/isTransientError' step.

  If an error occurs in the service logic, Integration Server exits the try sequence because it is set to exit on failure. As a result, Integration Server executes the catch sequence.

- **STEP 1.2 - Catch SEQUENCE exits on DONE**

  Integration Server only executes the catch sequence if the try sequence failed. The catch sequence contains logic that evaluates the error to determine whether the error is transient.

  Because the catch sequence exits on DONE, the sequence is successful after Integration Server executes all the steps in the sequence. After the catch sequence is successful, the outer sequence, which is set to exit on success, also exits. Integration Server executes the next step, which is the BRANCH on '/isTransientError' step.

- **STEP 1.2.1 - Catch the last error**

  To determine whether a transient error occurred, the catch sequence first invokes the pub.flow:getLastError service to catch the error that caused the try sequence to fail.

  Important:
  The pub.flow:getLastError service must be the first service invoked within the catch sequence. If it is not first and a preceding service in the catch sequence fails, the error thrown in the try sequence is overwritten with the new error.

- **STEP 1.2.2 - Determine whether the error is a transient error**

  This step evaluates the contents of the *lastError* document that the pub.flow:getLastError service returns to determine whether the try sequence failed because of a transient error. You might use multiple services or flow steps to determine whether a transient error occurred.

  Note:
  If the service logic in the try sequence includes an adapter service and a transient error occurs during adapter service execution, the adapter service throws an exception that extends the ISRuntimeException. Ensure that your catch sequence interprets the adapter service exception that signals retry. For more information, see "Requirements for Retrying a Service" on page 904.

- **STEP 1.2.3 - Set flag to indicate whether the service should retry**

  This step sets the transient error flag based on whether the try sequence failed because of a transient error. In this example if a transient error occurred, the variable *isTransientError*) is set to "true".

After this step executes, Integration Server exits the catch sequence, exits the outer sequence, and then executes the BRANCH on '/isTransientError' step.

- **STEP 2 - Check transient error flag**

  This step uses the value of *isTransientError* to determine whether the service should throw an ISRuntimeException.

  If the try sequence executed successfully, *isTransientError* is null. As a result, Integration Server falls through to the end of the service because the value of the switch variable does not match any of the target steps. Integration Server will not attempt to retry the service.

  If the try sequence failed, but the catch sequence determined that the error was not transient, the catch sequence does not set *isTransientError* to "true". It might be null or the catch sequence might set *isTransientError* to another value, for example, "false". Either way, Integration Server falls through to the end of the service because the value of the switch variable does not match any of the target steps. Integration Server will not attempt to retry the service.

  If the try sequence failed and the catch sequence determined that the error was transient, *isTransientError* is "true", and as a result, Integration Server executes the next step.

- **STEP 2.1 - Throws ISRuntimeException**

  Integration Server executes this step to invoke pub.flow:throwExceptionForRetry service when the value of *isTransientError* is "true". This service wraps the exception generated by the transient error in the try sequence and re-throws it as an ISRuntimeException.

  If the service is configured for retry, Integration Server retries the service if the maximum number of retries has not been reached. For more information, see "Configuring Service Retry" on page 179.

44 Creating Client Code

- Building a Java Client 910
- Building a C/C++ Client 912
- Building a Browser-Based Client 913
- Building a REST Client 919

Client code is application code that invokes a service on Integration Server. It typically performs the following basic tasks:

- Prompts the user for input values for the service that the client invokes (if the service takes input)
- Places the input values into an input document
- Opens a session on Integration Server
- Invokes the service
- Receives output from the service
- Closes the session on Integration Server
- Displays the service's output to the user

Using Designer you can automatically generate client code in Java and C/C++. The generated client code can serve as a good starting point for your own development.

You can also build client code on your own for browser-based clients and REST clients.

Note:
By default, Integration Server writes a response message to an HTTP client using the same content type that was included in the request message. However, you can specify that your service send the response message using a different format. For example, if your service receives a request that specifies text/xml as the content type, you can send a response message that specifies text/html for the content type. To change the content-type of the response message, code your service to call the pub.flow:setResponseHeader service before it writes output to the pipeline.

Building a Java Client

You can have Designer generate Java client code that invokes a service. The generated code includes a rudimentary user interface that uses the classes in the IS folder in which the Java service resides. It is not intended for use "as is" in custom applications. After generating the code, modify it as necessary. You can update the client code to:

- Invoke built-in services. For information about the services that are available, see the *webMethods Integration Server Built-In Services Reference*.
- Use the webMethods Integration Server Java API. For more information, see the *webMethods Integration Server Java API Reference*.

Limitations when Generating Java Client Code

- When Designer generates Java client code, it ignores input or output variables that are of type Object or Object list. Designer does not generate client code for these variables.
- When Designer generates Java client code, Designer replaces any space in a variable name with an underscore.

- The Java client code that Designer generates does not support multiple input or output variables with the same name.

If you want to override these limitations, you will need to modify the client code that Designer generates.

Files that Designer Generates for a Java Client

Designer generates the following files for a Java client application.

Readme.txt	A file that contains information and instructions for the Java client code. Read this file for information about compiling and running the Java client application.
ServiceName.java	An example file, encoded in ISO8859_1, that contains the application code for the Java client.

Generating Java Client Code

To generate Java client code that invokes a service

1. Open the service for which you want to generate client code by double clicking it in the Package Navigator view.

2. Right click in the editor to view the context menu, and select **Generate Code**.

3. In the Code Generation window, select **For calling this service from a client** and click **Next**.

4. In the **Language** field, select **Java**, and then click **Next**.

5. Specify the directory where you want Designer to place the generated client code.

   Either select an existing directory or type the path for a new directory. If you type the path for a new directory, Designer creates the directory.

6. Click **Finish**.

   Designer generates the file that contains the Java client code (*ServiceName*.java) and a Readme.txt file. To complete your client application, see the Readme.txt file located in the same directory as your client code.

   Note:
   If the client will connect to Integration Server using the Secure Socket Layer (SSL), in addition to following the instructions in the Readme.txt file, you must ensure that the unlimited strength jurisdiction policy files (local_policy.jar and US_export_policy.jar) are installed as part of your JVM. If you are using the JVM that was installed with Integration Server, no

further action is needed. If you are using a different JVM, obtain the files from the JDK provider.

Building a C/C++ Client

You can use Designer to generate C/C++ client code that invokes a service.

Prerequisites for Generating C/C++ Client Code

Before you can generate code for a C/C++ client, ensure:

- Integration Server is running and connected to Designer.
- A platform that has the C/C++ compiler (for example, GCC) is installed. Integration Server generates code for the following platforms: Windows, Solaris, HP-UX, Linux, AIX.
- The wm-isclient.jar file is in the classpath for Designer. The client.jar file is a webMethods file that is located in the *Software AG_directory* \common\lib directory.
- The Make facility is installed.
- JDK 1.1.x is installed (if you intend to use the C libraries provided with Integration Server and Designer).

Important:
The provided C libraries are built using JDK 1.1.7. If you want to use a different version of the JDK to compile C/C++ services, you need to rebuild the C/C++ libraries with that JDK and then replace the old library files with the rebuilt ones. For more information about rebuilding the C libraries, see the README installed with the C/C++ SDK. To rebuild the C libraries, you need use the C/C++ SDK. The C/C++ SDK is not installed by default. To install the C/C++ SDK, select it from the list of installable components during installation.

Limitations when Generating C/C++ Client Code

- When Designer generates C/C++ client code, it ignores input or output variables that are of type Object or Object list. Designer does not generate client code for these variables.
- When Designer generates C/C++ client code, it replaces any space in a variable name with an underscore.
- The generated C/C++ client code does not support multiple input or output variables with the same name.

If you want to override these limitations, you will need to modify the client code that Designer generates.

Files that Designer Generates for a C/C++ Client

Designer generates the following files for a C/C++ client application.

CReadme.txt	A file that contains information and instructions for the C client code. Refer to this file for information about compiling, running, and deploying your C/C++ client application.
ServiceName.mak	A file that contains compiler settings for the C/C++ client. Be sure to update this file with the correct settings for your environment.
ServiceName.c	An example file that contains the C/C++ client code. It is not intended for use "as is" in custom applications.

Generating C/C++ Client Code

To generate C/C++ client code that invokes a service

1. In the Package Navigator view, open the service for which you want to generate client code.
2. In the editor, right click the service to view the context menu, and select **Generate Code**.
3. In the Code Generation window, select **For calling this service from a client** and click **Next**.
4. In the **Language** field, select **the C/C++ platform for which you are creating client code** and then click **Next**.
5. Specify the directory where you want Designer to place the generated client code.

   Either select an existing directory or type the path for a new directory. If you type the path for a new directory, Designer creates the directory.
6. Click **Finish**.

   Designer generates the file that contains the C client code (*ServiceName*.c), a file that contains compiling settings (*ServiceName*.mak), and a CReadme.txt file.

   Modify the generated client code to meet your site's needs. You can update the client code to invoke built-in services and to use the webMethods C API. For information about the built-in services that are available, see the *webMethods Integration Server Built-In Services Reference*. For documentation about the C API, see *webMethods Integration Server C/C++ API Reference*.

   To complete your client application, refer to the CReadme.txt file located in the same directory as your client code.

Building a Browser-Based Client

Build a browser-based client by creating one or more web pages with embedded URLs that invoke services. You can use any tool you want to build the web pages.

When Integration Server receives the first URL to invoke a service from the browser-based client, it creates a session for the client on Integration Server. The session information is stored in a cookie in the browser. As the user of the browser-based client clicks on links to URLs that invoke services, Integration Server uses the cookies to find session information for the client. Integration Server keeps the session information for the client until the session expires. Sessions expire based on the configured session time-out value. For more information about setting the session time-out limit, refer to *webMethods Integration Server Administrator's Guide*

Note:
You cannot use Designer to generate browser-based clients.

Prerequisites for Building Browser-Based Client Code

Before you can build browser-based client code, ensure:

- Integration Server is running.
- The input values for each service you want the browser-based client to invoke are defined. You will need to include the input values in the URL that you use to invoke a service.

URL Client Uses to Invoke Services

When embedding an URL that invokes a service into a web page of a browser-based client, you can use either HTTP GET or HTTP POST. The URL for either method is similar to the following:

Item	Description
1	Identifies the Integration Server where the service to invoke resides.
2	Specifies the required keyword "`invoke`", which tells Integration Server that the URL identifies a service that is to be executed.
3	Identifies the folder in which the service to invoke resides. Separate subfolders with periods. This folder name is case sensitive. Be sure to use the same combination of upper and lower case letters as specified in the folder name on Integration Server.
4	Identifies the service that you want to invoke. The service name is case sensitive. Be sure to use the same combination of upper and lower case letters as specified in the service name on Integration Server.
5	Specifies the input values for the service. Specify a question mark (?) before the input values. The question mark signals the beginning of the *query string* that contains the input values. Each input value is represented as *variable=value*. The *variable* portion is case sensitive. Be sure to use the same combination of upper

Item	Description
	and lower case letters as specified in your service. If your service requires more than one input value, separate each *variable=value* with an ampersand (&).

Note:
Only specify the query string portion of the URL when using the HTTP GET method.

Note:
If you are serving the web pages that invoke services from an Integration Server, you can use a relative URL to invoke the service. By doing so, you can serve the exact web page from several servers without having to update the URLs.

HTTP GET Method

To use the GET method, embed a URL that includes all the input values for the service in the query string portion of the URL. When the server receives the URL, it translates the input values into an IData object. For more information about how the server creates the IData object that it sends to the service, see "How Input Values are Passed to the Service the Browser-Based Client Invokes" on page 916.

HTTP POST Method

To use the POST method, create an HTML form in your web page. Create fields in the HTML form in which a user will supply the input information. The values you specify for the NAME attributes of the HTML form fields should match the names of input values that the service expects. Be sure to use the exact combination of upper and lower case letters as specified in your service. For example, if your service requires the input values *sku* and *quantity*, you might create an HTML form with the following fields:

```
<SELECT NAME="sku">
<OPTION VALUE="A1">A1</OPTION>
<OPTION VALUE="B2">B2</OPTION>
<OPTION VALUE="C3">C3</OPTION>
</SELECT>

<INPUT TYPE="TEXT" NAME="quantity" VALUE="1">
```

Specify the URL for the service in the ACTION attribute and "POST" in the METHOD attribute. For example:

```
<FORM ACTION="/invoke/sample.webPageDemo/getProductCost" METHOD="POST">
```

After the user fills in the form and submits it, the web browser creates a document that contains the information the user supplied in the HTML form (performs an HTTP POST). The browser invokes the URL identified in the ACTION attribute, which invokes the service on Integration Server, and the browser posts the document that contains the user's input information to Integration Server. For more information about how the server creates the IData object that it sends to the

service, see "How Input Values are Passed to the Service the Browser-Based Client Invokes" on page 916.

How Input Values are Passed to the Service the Browser-Based Client Invokes

Regardless of whether a browser-based client uses a URL that uses HTTP GET or HTTP POST, when Integration Server receives the URL, it creates an IData object from the input information that the browser-based client supplies. Integration Server then passes the IData object to the specified service. This becomes the pipeline upon which the service operates.

To create the IData object, Integration Server creates two key/value pairs for each input value: one of type String and one of type String list. For example, the following URL specifies input values that contain the variable *sku* with value A1 and *quantity* with value 1, and the resulting IData object that Integration Server creates:

```
/invoke/sample.webPageDemo/order?sku=A1&quantity=1
```

Key	Value	Data Type
sku	A1	String
skuList	A1	String list
quantity	1	String
quantityList	1	String list

Note:
Avoid using input variable names that end in "List." Although Integration Server accepts variable names ending in "List," the resulting IData might not be structured in the way you need. For example, if you pass in a variable called *skuList*, the resulting IData contains a String called *skuList* and a String list called *skuListList*. Additionally, if you pass in variables named *sku* and *skuList*, subsequent *sku* and *skuList* variables in the query string might not be placed in the IData fields as expected.

If you must use "List" at the end of your variable name, consider using "list" (lowercase) or appending one or more characters at the end of the name (for example, abcListXX).

When Browser-Based Clients Pass Multiple Values for the Same Input Variable

When Integration Server receives multiple input values that are associated with the same variable name, the String variable in the IData object will contain only the value of the first variable. The String list variable will contain all the values. For example, the following shows a URL that contains two values for the variable *year* and the resulting IData object that Integration Server creates:

```
/invoke/sample.webPageDemo/checkYears?year=1998&year=1999
```

Key	Value	Data Type
year	1998	String
yearList	1998	String list
	1999	

Similarly, if the HTML form contains two fields with the same name and a user supplies values for more than one, the String variable in the IData object contains only the value of the first variable; the String list variable contains all values. For example, the following shows sample HTML code that renders check boxes:

```
<INPUT TYPE="checkbox" NAME="Color" VALUE="blue">Blue<BR>
<INPUT TYPE="checkbox" NAME="Color" VALUE="green">Green<BR>
<INPUT TYPE="checkbox" NAME="Color" VALUE="red">Red<BR>
```

If the browser user selects all check boxes, the document that is posted to Integration Server will contain three values for the variable named *Color*. The following shows the IData object that the server passes to the service:

Key	Value	Data Type
Color	blue	String
ColorList	blue	String list
	green	
	red	

When Browser-Based Clients Pass Multiple Input Variables with the Same Name

If the URL that a browser-based client passes to Integration Server contains multiple variables that have the same name, Integration Server determines how to handle the duplicate variables based on the setting of the watt.server.http.listRequestVars server configuration parameter.

To have Integration Server:

- **Create list variables for only duplicate variables**, set watt.server.http.listRequestVars server to `asNeeded`. This is the default.

  With this setting, Integration Server creates an IData object that contains:

  - String variable that contains the first occurrence of each input variable
  - String list variable that contains all occurrences of each duplicated variable

  For example, this request:

  ```
  /invoke/sample.webPageDemo/checkYears?year=1998&year=1999&month=June
  ```

  Integration Server creates the following IData object:

Key	Value	Data Type
year	1998	String
yearList	1998	String list
	1999	
month	June	String

- **Create list variables for all variables**, set watt.server.http.listRequestVars server to `always`.

  With this setting, Integration Server creates an IData object that contains:

  - String variable that contains the first occurrence of each input variable
  - String list variable that contains all occurrences of each input variable

  For example, for this request:

  ```
  /invoke/sample.webPageDemo/checkYears?year=1998&year=1999&month=June
  ```

  Integration Server creates the following IData object:

Key	Value	Data Type
year	1998	String
yearList	1998	String list
	1999	
month	June	String
monthList	June	String list

- **Create no list variables**, set watt.server.http.listRequestVars server to `never`.

  With this setting, Integration Server ignores duplicates of the same variable and creates an IData object that contains only a String variable that contains the first occurrence of each input variable.

  For example, for this request:

  ```
  /invoke/sample.webPageDemo/checkYears?year=1998&year=1999&month=June
  ```

Key	Value	Data Type
year	1998	String
month	June	String

- **Throw a ServiceException**, set watt.server.http.listRequestVars server to `error`.

With this setting, Integration Server throws a ServiceException if duplicates of the same variable are found.

For more information about this configuration parameter, see *webMethods Integration Server Administrator's Guide*.

How Integration Server Returns Output from the Service the Client Invoked

By default, when a service is invoked by a browser-based client, Integration Server displays the output from the service in an HTML web page, using a table to render the output values.

Alternatively, you can assign an output template to the service that the browser-based client invokes. In this case, Integration Server formats the output using the assigned output template. Using an output template gives you the opportunity to design how you want the output to display. With a template you can embed URLs that link to other resources or that invoke another service to perform the next step of the task that the browser-based client performs. You can use the results from one service to dynamically construct how the output is displayed and/or as input into a subsequent service that is invoked. For more information about output templates, see "About Service Output Templates" on page 195.

Building a REST Client

To interact with a REST application, a customer or partner must create a REST client that can send properly formed requests to the REST server and handle responses sent by the server.

For information about creating a REST resource, see "Working with REST" on page 473. For more information about how Integration Server uses REST and REST client requirements, see the *REST Developer's Guide*.

45 Comparing Integration Server Packages and Elements

- Working with the Compare Editor 922
- Comparing Packages and Elements 925

Designer provides you with the ability to compare packages and elements in Integration Server. The compare tool is useful to compare packages and elements on the same server or on different servers, and to track changes to a package or element during the development process. For example, you can use the compare tool to identify the differences between the development, staging, and production versions of a package or element. You can use the tool to compare:

- Packages
- Folders
- Flow Services
- Integration Server Document Types

The differences between the items that you compare are presented in a compare editor along with annotations to indicate the changes.

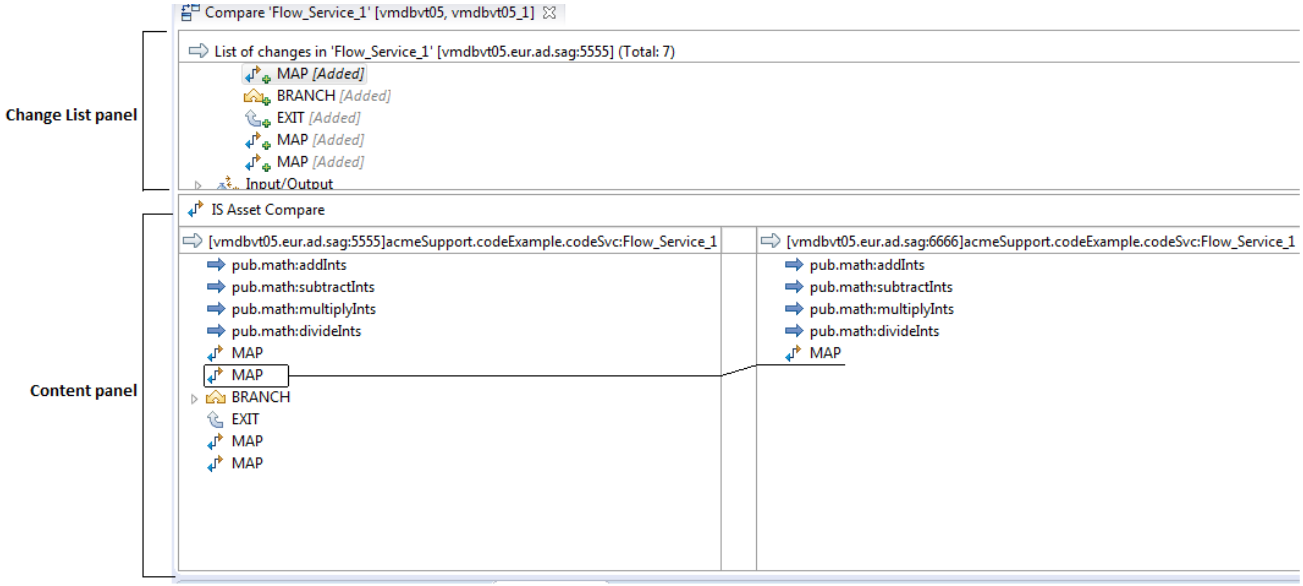
You can also use the compare tool to compare two revisions of an element in a local service development project. For information, see "Comparing Revisions of an Element" on page 116.

The ability to compare packages and elements is available only with Integration Server 9.9 and later.

Note:
To compare packages and folders, the Integration Server on which they are located must have the pub.assets:getChecksums service in the WmPublic package. For additional details on the pub.assets:getChecksums service, see *webMethods Integration Server Built-In Services Reference*.

Working with the Compare Editor

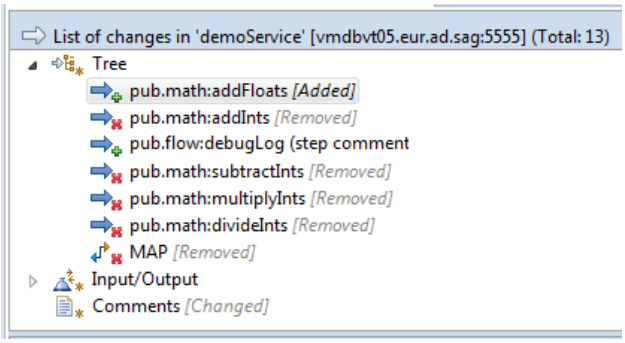
You can open the compare editor using the **Compare Element(s) With** command available when you right click on an element in the Package Navigator view. If you select one element in the tree, you can compare it with another similar element on a different Integration Server. You can also select two elements of the same type anywhere in the Package Navigator or on different servers and compare them with each other. Designer shows the differences between the two items that you compare in a compare editor as shown in the following example:



The compare editor, which is different from the element editor, consists of the following panels:

- **Change List Panel:** Shows, in the top panel, the list of differences between the packages or elements being compared.
- **Content Panel:** For flow services and IS document types, this panel, which appears below the **Change List** panel, provides a drill-down, visual view of the differences which are listed in summary form in the **Change List** panel. In the case of packages and folders, you can right click on a changed item in the **Change List** panel and select **Compare Contents** to open the element-level view. Designer opens a compare editor that shows the element level view of the changed item that you selected in a new tab.

Change List Panel

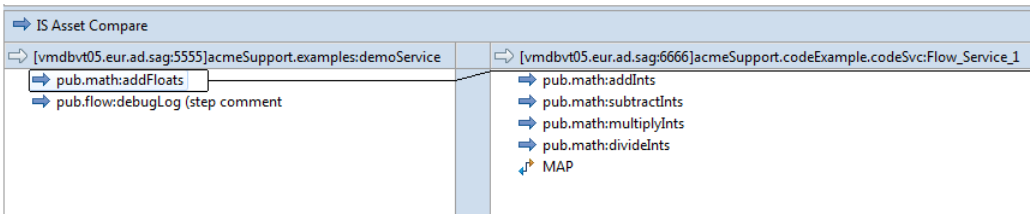


The **Change List** panel is the top panel in the compare editor. The **Change List** panel lists out the differences between the packages or elements that you compare in a tree structure. The header text in the panel shows the names of the two packages or elements that are being compared and the total number of changes. The changes are annotated with respect to the first element that you selected. The following annotations are used to indicate the differences:

- *Changed*- An item is present in both packages or elements being compared but has changed.

- *Added*- An item is present only in the first package or element being compared, and is not present in the second package or element.
- *Removed*- An item is present only in the second package or element being compared, and is not present in the first package or element.
- *Repositioned* (*x to y*) - An item has the position *x* in the second element being compared and the position *y* in the first element.

Content Panel



The **Content** panel is located below the **Change List** panel. An element-level visual view of the differences between the elements being compared is shown in the **Content** panel as described below:

- For a flow service or an IS document type, the difference that you select in the **Change List** panel is displayed in detail in the **Content** panel.

  Note:
  When you select a specific difference in a package or folder, Designer opens a new compare editor next to the original compare editor.

- Designer displays the first element on the left side of the **Content** panel and the second element on the right side. The path of the elements compared are displayed at the top of the panels respectively.
- Each change is indicated by highlighting an existing item in a package or element on one side with a box and using a line to link the item to the corresponding item in the other package or element on the other side, at the position where the item is present or should have been.
- Designer allows you to edit an element that you have locked from the **Content** panel. Right-click on the element and select **Open in Editor** to open the element in an editor.

  After you have made changes in the element, save the changes. Designer displays the **Reload Compare Editor** dialog box, prompting you to confirm a refresh in the compare editor. Click **OK** to refresh the compare editor with the changes to the element.
- Use the toolbar icons or their equivalent keyboard shortcuts listed below to navigate between the changed items:
  - Previous difference: `CTRL + ,` or
  - Next difference: `CTRL + .` or

- Use the toolbar icons listed below to merge the changes:
  - Merge changes from left to right:
  - Merge changes from right to left:

  A check mark appears after the changes are merged successfully.

Merging IS Elements

Before you perform the merge operation, you must ensure that you have write access to the element. The merge icons are enabled only if there are any changes. If the elements are read-only, the corresponding icons are disabled. For example, if the elements on the right side are read-only, the left to right merge icon is disabled. Changes cannot be merged if:

- IS element is not locked for edit
- IS element is retrieved from VCS repository
- IS element does not have Write ACL privilege
- Changes depend on some other conditions. For example in IS document type element, you cannot merge **Time to live** property if the **Discard** property is set to false. For more information on IS element properties, see "Properties" on page 953.

Comparing Packages and Elements

Keep the following points in mind when you use the compare tool:

- The results of the compare tool are useful only when you compare elements that are of the same type.
- When you compare two packages or elements, Designer opens the compare editor only if there are one or more differences between the items compared. If there are no differences to show, Designer displays a message to indicate this.
- You can only view differences, if any, using the compare editor. For details, see "Working with the Compare Editor" on page 922. If you want to modify a package, folder or any other element, use the navigator or editor provided for this purpose.
- You can only compare packages and assets from the same version of Integration Server. Comparison of assets across different versions of Integration Server is not supported. Internal changes in assets from one release to another might result in functionally identical assets being reported as different.

Comparing Integration Server Elements

To compare two IS elements

1. In Package Navigator view, select one of the following IS elements and hold the CTRL key while you select another package or folder with which you want to compare it.
   - Flow service
   - IS document type
   - JMS trigger
   - Adapter service
   - Adapter notification
2. To compare IS elements with each other, right-click, select **Compare Element(s) With > Each Other**.
3. To compare IS elements with the elements from another server listed in the context menu, right-click, select **Compare Element(s) With > Other Server**.
4. In the compare editor that Designer opens, double-click on a difference displayed in the **List of changes** panel to open up the element-level view in the Content panel below.

Comparing Integration Server Packages or Folders

To compare two Integration Server packages or folders

1. In Package Navigator view, select a package or a folder and hold down the CTRL key while you select another package or folder with which you want to compare it.
2. Right-click either selected item and select **Compare Element(s) With > Each Other**.
3. Alternatively, select a package or folder, right-click and select **Compare Element(s) With > Other Server**. Select the package or folder that you want to compare this one with from the servers listed in the context menu.
4. In the compare editor that Designer opens, select a difference in the **List of changes** panel, right-click and:
   a. Select **Show Left Element in Package Navigator** with the following results, depending on whether the item is shown as *Changed*, *Added*, or *Removed*:

- *Changed*: The corresponding item in the first package or folder is shown in the Package Navigator view.
- *Added*: This indicates that an item is present only in the first package or folder being compared, and selecting the **Show Left Element in Package Navigator** option will jump to the item in the Package Navigator view for the first package or folder.
- *Removed* This indicates that an item is present only in the second package or folder, and selecting the **Show Left Element in Package Navigator** option will take you to the first package or folder in the Package Navigator view from which the item was removed or under which it was expected to be present.

b. Select **Show Right Element in Package Navigator** with the following results, depending on whether the item is shown as *Changed*, *Added*, or *Removed*:

- *Changed*: The corresponding item in the second package or folder is shown in the Package Navigator view.
- *Added*: This indicates that an item is present only in the first package or folder being compared, and selecting the **Show Right Element in Package Navigator** option will take you to the second package or folder in the Package Navigator view from which the item was removed or under which it was expected to be present.
- *Removed*: This indicates that an item is present only in the second package or folder, and selecting the **Show Right Element in Package Navigator** option will jump to the item in the Package Navigator view for the second package or folder.

c. Select **Open in Compare Editor** to open the element-level view of the difference in another instance of the compare editor.

> **Note:**
> The **Open in Compare Editor** option in only available for *Changed* items.

5. For packages, in the **List of changes** panel, select the required property under **Properties**. The compare editor displays the comparison of properties in the **IS Asset Compare** panel.

> **Note:**
> Properties are compared only for packages, and not for folders.

46 Connecting to webMethods API Portal for Publishing REST API Descriptors

- Configuring a Connection to API Portal 930

Designer helps you publish REST API descriptors created on Integration Server to API Portal. Before you can publish the REST API descriptors, you must configure the required connection to API Portal.

Configuring a Connection to API Portal

You need to provide specific information in Designer for initiating a connection to API Portal. Designer saves this information in a connection configuration. You can add, edit, and update connection configurations in Designer.

Before you configure a connection to API Portal, ensure that the following criteria are met:

- API Portal is already configured and information about the host address and the port on which API Portal is running are available.
- A valid user account is created on API Portal.
- The tenants for which the REST API descriptors are to be published are already created on API Portal. Every tenant must have a unique connection configuration.

Note:
For information about publishing REST API descriptors to API Portal after configuring the required connection, see "Publishing REST API Descriptors to API Portal" on page 500.

Adding a Connection Configuration for API Portal

The number of connections to API Portal that you need to configure depends on the number of different tenants for which the REST API descriptors are to be published. For every tenant, you must create a unique connection configuration.

To add a connection configuration for API Portal

1. In Designer, select **Window > Preferences**.
2. In the preferences navigation tree, select **Software AG > API Portal**.
3. Click **Add**.
4. In the **Add Connection Configuration** dialog box, enter the following information:

Field	Description
Name	The name to use for the connection configuration. **Note:** The name cannot contain control characters, special characters, and characters outside of the

Field	Description
	basic ASCII character set, such as multi-byte characters.
Host	The host name of API Portal for the connection configuration.
Port	The port number on API Portal for the connection configuration.
User	The name of a valid user account on API Portal.
Password	The password for the specified **User**. If required, you can omit the password at this time. In such a situation, API Portal prompts for the password each time your user authorization is required for connecting to API Portal.
Save password (in the Eclipse secure storage)	Indicates whether the password for the specified user account should be saved in Eclipse secure storage. API Portal uses this password from the Eclipse secure storage whenever user authorization is required. If you want to save the password in Eclipse secure storage, select this check box. If you decide not to save the password in Eclipse secure storage, you must specify your password each time your user authorization is required for connecting to API Portal.
Tenant	The tenant for which the REST API descriptors are to be published.
Secure connection	Indicates whether the session for connection to API Portal is through HTTP or HTTPS. If you want to open an HTTPS session by using the Secure Socket Layer (SSL), select this check box. If you want to open an HTTP session, clear this check box.

5. To verify whether API Portal can be accessed by using the specified information, click **Test**.

6. To store the connection configuration details, click **OK**.

   A connection configuration is added to the Connections page with the specified details. The first connection configuration that you create is automatically marked as default. This default configuration is indicated with a check mark on the Connections page. Designer always uses the default connection configuration for API Portal.

   Note:

You can change the default connection configuration for API Portal. For more information, see "Changing the Default Connection Configuration for API Portal" on page 933.

Editing a Connection Configuration for API Portal

You can edit a connection configuration for API Portal if there are any changes in the configuration values.

To edit a connection configuration for API Portal

1. In Designer, select **Window > Preferences**.
2. In the preferences navigation tree, select **Software AG > API Portal**.
3. Click **Edit**.
4. Enter new values in the connection configuration fields you want to change.
5. In the **Edit Connection Configuration** dialog box, click **OK**.
6. In the **Connections** page, click **OK**.

Removing a Connection Configuration for API Portal

You can remove connection configurations for API Portal one at a time from the **Connections** page. You *cannot* remove the default connection configuration.

To remove a connection configuration for API Portal

1. In Designer, select **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > API Portal**.
3. Select the API Portal connection configuration you want to remove.
4. Click **Remove**.

   Designer prompts you to confirm that you want to remove the selected connection configuration.
5. Click **OK**.

Changing the Default Connection Configuration for API Portal

When connecting to API Portal, Designer always uses the default connection configuration. Therefore, a connection configuration must always be marked as default on the Connections page.

You might change the default connection configuration when you want to publish REST API descriptors for a tenant which is different from that specified in the default configuration.

To change the default connection configuration for API Portal

1. In Designer, select **Window > Preferences**
2. In the preferences navigation tree, select **Software AG > API Portal**.
3. Select the **Default** check box for the connection configuration you want to be the default.

   Designer prompts you to confirm that you want to replace the existing default with the new one.
4. Click **OK**.

47 API Portal Preferences

Field	Description
Name	The name to use for the API Portal connection configuration.
Host	The host name of API Portal for the connection configuration.
User	The name of a valid user account on API Portal.
Tenant	The tenant for which the REST API descriptors are to be published.

48 Document Expansion Preferences

On the Document Expansion preferences page, specify how Designer displays and expands the content of document variables.

Field	Description
Document expansion level	Specifies the depth to which Designer expands a nested document automatically. Designer does not expand the contents of document variables nested deeper than the specified level. The minimum is 1 level. The maximum is 100 levels. The default is 10 levels.
Recursive document expansion level	For a recursive document (one that contains a reference to itself), specifies the number of instances of that recursion that Designer expands automatically. The minimum is 1 level. The maximum is 100 levels. The default is 5 levels. The specified level must be less than or equal to the level for **Document expansion level**.
Variables to expand per document	Specifies the number of child variables that Designer displays automatically for each document variable. The minimum is 1 level. The maximum is 100 levels. The default is 25 variables. This preference applies when entering values for a document variable only.

Note:

If the service signature contains very complex document structures where the documents are nested and deep, Software AG Designer may stop responding if you try to expand those complex document structures, with the default document expansion settings.

If you are developing services for connectors with signatures having complex document structures where the documents are nested and deep, the recommended document expansion setting (**Windows Preferences > Software AG > Document Expansion**) values are as follows:

- Document expansion level = 2

- Recursive document expansion level = 1
- Variables to expand per document = 10

For information on how to create connectors with signatures having complex document structures where the documents are nested and deep, use the hybrid approach as described in the *Administering webMethods CloudStreams* document.

49 Integration Server Preferences

Field	Description
Name	Name to use for this Integration Server.
Host Name or IP Address	Host name (for example, `workstation5.webmethods.com`) or IP address (for example, `132.906.19.22` or `2001:db8:85a3:8d3:1319:8a2e:370:7348`) of the Integration Server to which Designer is to connect.
Port	Port number on which Integration Server listens for requests.
Default	By default, a new Designer installation includes a server definition named Default. This server is marked as the default server and is configured to use *localhost:5555*. If a user creates or edits a process and no server definitions are connected, Designer automatically connects to the default server definition. Refer to the *webMethods BPM Process Development Help* for more information.
Status	Indicates whether or not Designer is connected to the Integration Server specified in this definition. Possible statuses are **Connected**, **Not connected**, **No userid and password**. To connect to an Integration Server that has the **No userid or password** status, click select the associated definition, click **Edit**, and enter the user ID and password information.
Offline	Specifies whether Designer is to attempt to connect to this Integration Server. By default, if a Process Development user creates or edits a process when no server definitions are connected, Designer will automatically try to connect to a server. Often, when connecting to a server, Designer will prompt the user to enter credentials. To prevent Designer from repeatedly prompting for credentials, you can place the server definition *offline*. Refer to the *webMethods BPM Process Development Help* for more information.

50 Service Development Preferences

- Adapter Service/Notification Editor Preferences 942
- Compare Editor Preferences 942
- Element Property Templates Preferences 943
- Flow Service Editor Preferences 944
- HTML Generation Preferences 945
- Java/C Service Editors Preferences 945
- Launching Preferences 946
- Local Service Development Preferences 946
- Package Navigator Preferences 947
- Publishable Document Type Preferences 949
- Results View Preferences 949
- Run/Debug Preferences 949
- Schema Editor Preferences 950
- Swagger Syntax Coloring Preferences 950
- Web Service Descriptor Editor Preferences 951

On the Service Development Preferences page, you can specify the behavior of editors and views in the Service Development perspective. You can also use the Service Development Preferences page to define property values and launching preferences for elements.

You can open the Service Development Preferences page by selecting **Window > Preferences** and then selecting **Software AG > Service Development** from the navigation tree.

Preference	Description
Show variables with fixed values	When selected, Designer displays the variables with fixed default values, which are hidden by default. You cannot override the default values assigned to these variables by mapping it to another variable or by assigning any input values to this variable during service execution. When the **Show variables with fixed values** property is selected, Designer displays these variables in the content and structure of service signatures, document and pipeline contents, and in the Run Configurations, Enter Input for *serviceName*, and Enter Input for *variableName* dialog boxes.

Adapter Service/Notification Editor Preferences

On the Adapter Service/Notification Error preferences page, you can specify whether data is automatically validated upon entry and whether metadata is automatically reloaded.

Preference	Description
Automatic data validation	When selected, value checking is performed on all user input. This safeguard makes sure that all user data are valid against current resource data.
Automatic polling of adapter metadata	When selected, Designer will reload metadata from the adapter every time it creates a new adapter service/notification. This option can be useful for adapter developers that are working on designing the metadata.
Use grouping in tree browser	Indicates whether tree structures for adapters will group items together. This may improve performance and may make it easier to locate items in tree browsers. If you selected the **Use grouping in tree browser** check box, in the **Limit visible items per group to** field, specify the maximum number of items that Designer groups together.

Compare Editor Preferences

Use the Compare Editor preferences page to customize settings for the compare editor.

Preference	Description
Do not include internal properties of element	Select this option to exclude internal properties and supporting files associated with elements being compared.
Show the change on single-click	Select this option to display the element-level view of a change on single-click.
Show Status Bar messages as tooltip	Select this option to display the status of an item as tooltip.
Reload compare editor	Select one of the following options to reload the change list after you merge and save an item: ■ **Prompt**: Displays a dialog box prompting to confirm the compare editor reload after an item is merged and saved. ■ **Never**: Never reloads the compare editor after an item is merged and saved. ■ **Always**: Automatically performs the compare editor reload after every merge and save operation.

Element Property Templates Preferences

Use the Element Property Template preferences page to define sets of property values for elements. You can create a template for a particular element and apply the template when creating new instances of the element. You can create multiple templates for an element type.

Note:
You can create property templates for flow, C/C++, and Java services.

Preference	Description
Templates list	Displays all the available property templates. Use this list to define new property templates and to view, edit, and remove the available property templates. You can also share the property templates across different instances of Designer, through the import and export operations.
Template properties	Displays the property names and values as defined in the template.

Note:
You will not be able to specify values for properties that must be unique for each element such as **Universal name** and **Output template** when defining templates.

Flow Service Editor Preferences

Use the Flow Service Editor preferences page to customize settings for working in the Flow service editor.

Preference	Description
Services List	Use this list to specify the list of services that appear under Insert on the flow service editor Palette view. Each row in the list represents a single command on the menu. Commands will appear on the menu in the order you specify them. You may add as many services as you need. The **Name** column specifies a label for the service. The **Service** column specifies the services associated with the labels on the menu.
Number of recently used services to list	Indicates the maximum number of recently-accessed services that you want to appear in the Palette view.
Validate flow service	When **Validate service references while saving** is cleared, Designer does not validate the referenced services while saving a flow service. By default, this check box is not selected. When **Validate service references while saving** is selected, Designer validates all the referenced services while saving a flow service. **Note:** Clearing this preference improves the performance of Designer while saving a flow service.
Grid Properties	When **Enable Grid** is selected, Designer displays a grid on the Layout tab of the flow service editor. Use **Grid Width** and **Grid Height** to specify the size of the grid.
Label Properties	In the Layout tab, specifies the height and width used for displaying the name of the service for an INVOKE step.
Default Pipeline Tree Tooltip	Specifies the tooltip that appears for pipeline variables. Select one of the following:

Select...	To...
Show variable path	Display the variable path as the tooltip
Show variable comment	Display the comment saved with the variable as the tooltip

Preference	Description	
	Show all variable properties	Display all the variable properties as the tooltip
Document references in Pipeline view	When **Show referenced document type name in Pipeline view** is selected, Designer displays the full namespace path of the referenced document types in Pipeline view. **Note:** You can also set this property by right-clicking anywhere inside the Pipeline view and selecting **Show Referenced Document Type Name**.	
Scrolling in Pipeline view	When **Enable independent scrolling** is selected, Designer enables the **Pipeline In**, **Pipeline Out**, **Service In**, **Service Out**, and **Transformers** columns to be scrolled horizontally and vertically independent of each other. This makes it easy to scroll through data when mapping a large amount of data in the Pipeline view. To hide the horizontal and vertical scroll bars, clear the **Enable independent scrolling** check box.	

HTML Generation Preferences

Use the HTML Generation preferences page to customize the contents of the HTML generated for an IS element.

Preference	Description
Display Properties	When selected, Designer includes the properties for the document type and the variables in the document type in the HTML generated for the document type.

Java/C Service Editors Preferences

Use the Java /C Service Editors preferences page to customize settings for working in the Java and C service editors.

Preference	Description
Protected line background color	Indicates the color to shade the protected sections of the Java service and the C/C++ service on the **Source** tab of the Java or C/C++ service editor. The Preferences window displays the current color on a button. To change the color: 1. Click the button to display the Color window.

Preference	Description
	2. Select a new color. 3. Click **OK**.
Default Java service signature	Indicates whether you want to use an IData signature or a Values signature for new Java services. Select: ■ **Use new Java service signature** to use an IData signature. ■ **Use old Java service signature** to use a Values signature. This type of signature was used for Integration Server version 3.5 and earlier.

Launching Preferences

Use the Launching preferences page to indicate whether Designer should save any open elements with unsaved changes before starting a launch configuration.

Preference	Description
Save required dirty editors before launching	Indicates whether Designer prompts you to save any elements with unsaved changes before starting a launch configuration. Specify one of the following: ■ **Always**. Designer saves any elements with unsaved changes automatically and does not prompt you to save any elements. ■ **Never**. Designer does not save any elements with unsaved change nor does Designer prompt you to save any elements. ■ **Prompt**. Designer prompts you to save any elements with unsaved changes.

Local Service Development Preferences

Use the Local Service Development Preferences to specify preferences related to local service development.

Preference	Description
Automatically reload IS assets	When this option is selected, Designer automatically reloads a package after you perform any VCS operations on the package. Designer reloads

Preference	Description
after VCS operations	the package on the Integration Server assigned as the local development server. This option is selected by default.
Move project to Integration Server package as linked resource	When this option is selected, Designer allows you to set any repository on your workspace as the local repository directory and a linked directory is created under the *Integration Server_directory* \instances\default\packages directory.
Enable local development in webMethods Microservices Runtime	When this option is selected, Microservices Container hosted on Docker can be used as local development server.
TCP Connection	Indicates the TCP URL of the local Docker daemon that listens for the Docker Remote API requests.
Timeout	Specifies the maximum number of seconds that Designerwaits for a response for TCP connection.

Package Navigator Preferences

Use the Package Navigator preferences page to specify safeguards when moving, renaming, and deleting elements in the Package Navigator view and to indicate the number of elements Designer caches for each session.

Preference	Description
Confirm before deleting	When selected, instructs Designer to notify you before deleting an element used by other elements, such as flow services, IS document types, specifications, or triggers. If Designer finds elements that depend on the element being deleted, Designer lists those dependents and prompts you to delete the element anyway or cancel the action. If you clear this check box, Designer deletes the element without prompting you.
Prompt before updating dependents when renaming/moving	When selected, instructs Designer to alert you when dependents exist. Dependents are elements that use the selected element, such as flow services, IS document types, specifications, or triggers. ■ If dependents exist, Designer lists those dependents before renaming or moving the selected element and prompts you to:

Preference	Description
	■ Rename/move the selected element and update dependent elements with the new name and location. ■ Rename/move the selected element only. ■ Cancel the action. If you clear this check box, Designer automatically updates dependents without prompting you.
Update local references when pasting multiple elements	When selected, Designer updates local references when copying and pasting a group of elements. When two elements within a group refer to each other, it is called a local reference. If you clear this check box, Designer retains the original references in the copied elements.
Automatically unlock upon save (non-VCS servers only)	When selected, Designer automatically unlocks elements after you save changes to them. This prevents you from forgetting to unlock elements; however, it may not be the best option if you save periodically while editing an element. (Not applicable for Java/C services.) **Note:** Do not select this option if the VCS Integration feature is enabled; it will cause synchronization problems with the VCS repository. **Note:** This option does not apply to local development servers.
Hide generated flow services	When selected, Designer does not display flow services automatically generated by a process in the Package Navigator view.
Verify hostname for server connections	When selected, Designer does not allow multiple connections to the same server (same IP address and same port) using different hostnames.
Number of elements to cache	Specifies the number of elements that you want to cache per Designer session. The higher the number of elements, the more likely an element will be in the cache, which reduces network traffic and speeds up Designer by caching elements that are frequently used. The total number of cached elements includes elements on all the servers to which you are connected. Click **Clear Cache** to remove all cached elements from memory. Clearing the cache does not remove flow services with breakpoints, flow services that are currently being

Preference	Description
	debugged, and unsaved elements. Keep in mind that the cache is automatically cleared when you close Designer or when you refresh the session.
Reset Tip Dialogs	Re-enables message boxes and reminders that have been disabled with the **Don't Show This Again** check box.

Publishable Document Type Preferences

Preference	Description
Display timeout message	When selected, Designer displays a message if the timeout limit for deliver and wait or publish and wait request elapses before Designer receives a response.

Results View Preferences

Use the Results view preferences page to specify the number of results to be displayed in Results view.

Preference	Description
Number of results to display	Specifies the number of results that you want Designer to display in the Results view. Set this preference to an integer smaller than 100. The default is 5. **Note:** If you are executing large documents multiple times, Software AG recommends that you set the **Number of results to display** option in the Results View preferences page to a smaller value, preferably 1, to ensure that the performance is not affected.

Run/Debug Preferences

Use the Run/Debug preferences page to customize the settings while running or debugging a service.

Preference	Description
Always show the No input dialog	When selected, Designer displays the No input dialog box, every time Designer runs a service with no input parameters.

Preference	Description
	When this check box is cleared, Designer no longer displays the No input dialog box when executing a service with no input parameters. This option is selected by default. **Note:** You can clear the **Always show the No input dialog** preference by selecting the **Do not show this dialog again** check box in the No input dialog box.
Flow Annotation Types	Indicates how the selected annotation, **Debug Current Instruction Pointer** or **Debug Call Stack**, appear in flow steps. You can choose to highlight the annotations or enclose the annotations in a box or dashed box in colors of your choice.

Schema Editor Preferences

Use the Schema Editor Preferences to control how deeplyDesigner expands the contents of an IS schema automatically.

Preference	Description
Schema expansion level	Specifies the depth to which Designer expands the contents of an IS schema automatically when you right-click a component and select **Expand**. For complex types and element declarations that are recursive, this preference can prevent Designer from expanding the same set of data infinitely. Set this preference to an integer greater than or equal to 1 but less than or equal to 9999. The default is 10 levels.

Swagger Syntax Coloring Preferences

Use the Swagger Syntax Coloring Preferences to set the font color and styles of tokens in a Swagger document.

Preference	Description
Token Styles	Specifies font color and styles of tokens in a Swagger document displayed in the Swagger tab of REST API descriptor editor. For each token, you can change the font, font color, background, and font style. **Note:**

Preference	Description
	In Eclipse context, a document is divided into a set of tokens, each describing a section of the document that has a different foreground, background, or font style.

Web Service Descriptor Editor Preferences

Use the Web Service Descriptor Editor Preferences to specify the encoding Designer uses when creating a web service descriptor from a WSDL with a URL that contains characters that do not comply with the URL Specification.

Preference	Description
Encoding for WSDL URL	Specifies the encoding that Designer uses when creating a consumer web service descriptor or a WSDL first provider web services descriptor from a WSDL whose URL contains special characters. If you select the **Encoding for WSDL URL** check box, do one of the following: ■ To use the default platform encoding, select **Default**. ■ To specify an encoding other than the platform default, select **Other**. Then select the encoding from the list next to **Other**.

51 Properties

- Integration Server Properties 955
- Package Properties 957
- Element Properties 962
- Document Type Properties 964
- Flat File Dictionary Properties 970
- Flat File Element Properties 970
- Flat File Schema Properties 984
- JMS Trigger Properties 987
- Link Properties 997
- OData Service Properties 998
- REST V2 Resource Properties 1003
- REST API Descriptor Properties 1003
- Schema Properties 1006
- Schema Component Properties 1007
- Service Properties 1022
- Specification Properties 1036
- Transformer Properties 1037
- Variable Properties 1038
- Web Service Connector Properties 1042

- Web Service Descriptor Properties 1047
- webMethods Messaging Trigger Properties 1060
- WebSocket Endpoint Properties 1067

Integration Server property information is available from the Service Development > Package Navigator view of Designer.

Use the Properties dialog box to view and edit properties for Integration Servers and packages. You can also use the Properties dialog box to view general information and permissions for Integration Server elements such as document types, services, flow steps, JMS triggers, web service connectors, and web service descriptors.

You can open the Properties dialog box by selecting the server, package, or element in Package Navigator view and selecting **File > Properties**. You can also open the Properties dialog box by right-clicking the server, package, or element and selecting **Properties**.

Integration Server Properties

You can view information about an Integration Server and assign event subscribers, permissions, and unlock elements on the server in the Properties dialog box.

To open the Properties dialog box, click the Integration Server in the Package Navigator of Designer and select **File > Properties**.

Event Manager Properties

Use the Event Manager page to subscribe to an event and to view, edit, suspend and/or delete an existing event subscription on the current server.

To open this page, select **File > Properties > Event Manager**.

Property	Description
View event subscribers for	Specifies the type of event whose subscriptions are displayed in this page. Select the event type for which you want to add, edit, or delete a subscription. The table in this page displays subscribers to the selected event type as follows:

Property	Description
Service	The fully qualified name of the subscriber (the event handler that executes when the event occurs). When you add a subscription, you can use the browse button to select the service.
Filter	A pattern string to further specify the events this event handler subscribes to. The information for which you create a filter depends on the event type you are subscribing to. The asterisk (*) character represents any string of characters and is the only wild-card character allowed in the pattern string. All other characters are treated literally. Pattern strings are case sensitive.

Property	Description	
	Comment	An optional comment describing this subscription.
	Enabled	Whether this subscription is currently active or inactive. Set this parameter to **true** to activate the subscription. Set this parameter to **false** to deactivate a subscription.

You use the buttons in this page to add, edit, and delete a subscription

Use this button...	To...
	Add a subscription.
	Insert a blank row for a subscription.
	Delete the selected subscription.
	Edit the selected subscription.

My Locked Elements

Use the My Locked Elements page to unlock elements for the selected server.

To open this page, in Designer select **File > Properties > My Locked Elements**.

Property	Description
Select the Elements to Unlock	Select the elements that you want to unlock. CTRL+click to select more than one or click **Unlock All** to unlock all of the elements in the list.

Server ACL Information

Use the ACL Information page to list the Access Control Lists (ACLs) contained on the Integration Server to which you are connected.

To open this page, select **File > Properties > Permissions**. This information is read only; to edit ACLs, users, and groups, use the Integration Server Administrator.

Property	Description
ACLs	The ACLs defined on the Integration Server to which you are connected. These include the default ACLs that were installed with the server. To edit an ACL, use the Integration Server Administrator.

Property	Description
User Group Association for '[ACL name]'	■ **Allowed.** The user group(s) that have been explicitly allowed to access the packages, folders, services, or other elements associated with this ACL. To edit a user group, use the Integration Server Administrator. ■ **Denied.** The user group(s) that have been explicitly denied access to the packages, folders, services, or other elements associated with this ACL.
Resulting Users for '[ACL name]'	Displays the names of users that the ACL authorizes, given the current settings in the Allowed and Denied lists. The server builds this list by looking at the groups to which each user belongs and comparing that to the groups to which the ACL allows or denies access. For details on how the server determines access, see *webMethods Integration Server Administrator's Guide*.

Server Information

View general information about a server from the Server Information page. To open this page, select **File > Properties > Server Information**.

Property	Description
Type	Type of server. This will be Integration Server.
Name	Name assigned to the Integration Server.
Location	Host name and port address of the Integration Server
User Name	The user name you use to connect to this Integration Server.
Connected	Indicates whether or not you are currently connected to this Integration Server.
VCS Enabled	Indicates whether this Integration Server is configured to use a version control system (VCS).
Proxy	Specifies the proxy server as set on **Window > Preferences > General > Network Connections**.

Package Properties

Use the Properties dialog box to view information about packages on the Integration Server and to assign package dependencies, permissions, replication services, startup and shutdown services.

To open the Properties dialog box, click the package in the Package Navigator of Designer and select **File > Properties**.

Package Information

The Element Information page displays the type and name of the Integration Server package.

To open this page, click the package in the Package Navigator of Designer and select **File > Properties > Element**.

Package Dependencies

The Package Dependencies page displays the packages on which this package is dependent. For example, if a package needs the services in another package to load before it can load, you need to set up package dependencies. You might also want to identify package dependencies if a startup service for a package invokes a service in another package. The startup service cannot execute if the package containing the invoked service has not yet loaded.

To open this page, click the package in the Package Navigator of Designer and select **File > Properties > Package Dependencies**.

Property	Description
Package	The name of the package you want webMethods Integration Server to load before the package selected in Package Navigator.
Version	The version number of the package you want loaded. More than one version of the same package might contain the services and elements that a dependent package needs Integration Server to load first. A dependency declared on a version is satisfied by a package with a version that is equal to or greater than the specified version. For example, to specify versions 3.0 or later of a package, type `3.0` for the version number. To specify versions 3.1 or later, type `3.1.0` for the version number. You can also use an asterisk (*) as a wildcard in the version number to indicate that any version number equal to or greater than the specified version will satisfy the package dependency. If any version of the package satisfies the package dependency, type `*.*` as the version number.

You use the buttons in this page to add, edit, and delete a package dependency.

Use this button...	To...
	Add a package dependency.
	Insert a blank row for a package dependency.
	Delete the selected package dependency.
	Edit the selected package dependency.

Package Settings

The Package Settings page displays general information about a package including package and JVM versions, build and patch numbers, publishers, and patch history.

To open this page, click the package in the Package Navigator of Designer and select **File > Properties > Package Settings**.

Property	Description
Package version	Specifies the version number for the package. Version numbers need to be in one of the following formats: *X.x* or *X.x.x* (for example, 1.0, 2.1, 2.1.3, or 3.1.2). By default, Designer assigns the version number 1.0 to a new package.
Build	Displays the build number of the package. The build number is a generation number that a user assigns to a package each time the package is regenerated. For example, a user might generate version 1.0 of the "Finance" package ten times and assign build numbers 1,2,3…10 to the different generations or builds of the package. The build number is not the same as the package version number. One version of a package might have multiple builds.
Description	Displays a brief description of the package written by the user who created the package release.
JVM version	Displays the version of the JVM (Java virtual machine) required to run the package.
Publisher	Displays the name of the publishing server that created the package release.
Patch number	Displays the patch numbers included in this release of the package.
Patch history	Displays a list of all the patches installed for this package release. When the server administrator installs a full release of the package (a release that includes all previous patches for the package), Integration Server removes the existing patch history. This helps the server administrator avoid potential confusion about version numbers and re-establish a baseline for package version numbers.

Property	Description
Name	The name of the package.
Version	The version number assigned to the package.
Build	The build number of the package. The build number is a generation number that a user assigns to a package each time the package (either full release or a patch) is regenerated.
Description	A brief description of the package or patch written by the user who created the package release.
Time	The time at which the package release (patch) was created.
JVM Version	The version of the JVM (Java virtual machine) required to run the package.

Property	Description
Publisher	The name of the publishing server that created the package release.
Patch Number	The patch numbers included in this release of the package.

Package Permissions

You assign an ACL to an element in the **Permissions** page of the Properties dialog box. Depending on the element you select, certain access levels are displayed. For example, for a package, you can only set List access. For details about the different levels of access available for elements, see *webMethods Integration Server Administrator's Guide*.

To open this page, click the package in the Package Navigator of Designer and select **File > Properties > Permissions**.

Property	Description
List ACL	Users in the Allowed list of this assigned ACL can see that the element exists and view the element's metadata (input, output, etc.).

Package Replication Services

The Replication Services page of the Properties dialog box specifies the services assigned as replication services for the package. A replication service is one that the webMethods Integration Server automatically executes when you create a release of a package (full or partial) or when you create an archive for a package. Replication services provide a way for a package to persist state or configuration information so that these are available when the published package is activated on the remote server.

To open this page, click the package in the Package Navigator of Designer and select **File > Properties > Replication Services**.

You use the following buttons in the Replication Services page to add, edit, and remove replication services.

Use this button...	To...
	Add a replication service.
	Insert a blank row for a replication service.
	Delete the selected replication service.
	Edit the selected replication service.

Package Startup/Shutdown Services

You use the Startup/Shutdown Services page of the Properties dialog box to add or remove the services that you want webMethods Integration Server to automatically execute when it loads or unloads a package into or from memory.

To open this page, click the package in the Package Navigator of Designer and select **File > Properties > Startup/Shutdown Services**.

Property	Description
Startup services	Displays the list of services that can be used as startup services and the list of assigned startup services in the package. A startup service is one that the webMethods Integration Server automatically executes when it loads a package into memory. Startup services are useful for generating initialization files or assessing and preparing (for example, setting up or cleaning up) the environment before the server loads a package. However, you can use a startup service for any purpose. For example, you might want to execute a time-consuming service at startup so that its cached result is immediately available to client applications. **Tip:** If a startup service invokes a service in another package, make sure to identify the other package as a package dependency for the package containing the startup service.
Available services	Displays a list of the services that can be used as startup services for the package. Any service in the package can be a startup service. After you select a service as a startup service, the service does not appear in the **Available services** list.
Selected services	Displays the startup services for the package. You use the following buttons under **Startup services** to add and remove startup services. **Use this button... / To...** Add the service selected in **Available services** as a startup service for the package. Remove the service selected in **Selected services** as a start up service for the package.
Shutdown services	Displays the list of services that can be used as shutdown services and the list of assigned shutdown services in the package. A shutdown service is one that the webMethods Integration Server automatically executes when it unloads a package from memory. Shutdown services are useful for executing clean-up tasks such as closing files and purging temporary data. You could also use

Property	Description
	them to capture work-in-progress or state information before a package unloads.
Available services	Displays a list of the services that can be used as shutdown services for the package. Any service in the package can be a shutdown service. After you designate a service as a shutdown service, the service does not appear in the **Available services** list.
Selected services	Displays the shutdown services for the package. You use the following buttons under **Shutdown services** to add and remove startup services.

Use this button...	To...
	Add the service selected in **Available services** as a shutdown service for the package.
	Remove the service selected in **Selected services** as a shutdown service for the package.

Element Properties

Use the Properties dialog box to view information about any Integration Server element listed in the Package Navigator. Integration Server elements include folders, subfolders, document types and services.

To open the Properties dialog box, click any Integration Server element in the Package Navigator of Designer and select **File > Properties**.

Element Information

The Element Information page displays the type and name of the Integration Server element.

To open this page, click the element in the Package Navigator of Designer and select **File > Properties > Element Information**.

Element Permissions

You assign an ACL to an element in the **Permissions** page of the Properties dialog box. Depending on the element you select, certain access levels are displayed. For example, for a package, you can only set List access. For details about the different levels of access available for elements, see *webMethods Integration Server Administrator's Guide*.

The ACLs assigned to an element are mutually exclusive; that is, an element can have different ACLs assigned for each level of access.

To open this page, click the element in the Package Navigator of Designer and select **File > Properties > Permissions**.

Property	Description
List ACL	Users in the Allowed list of this assigned ACL can see that the element exists and view the element's metadata (input, output, etc.).
Read ACL	Users in the Allowed list of this assigned ACL can view the source code and metadata of the element.
Write ACL	Users in the Allowed list of this assigned ACL can lock, edit, rename, and delete the element.
Execute ACL	Users in the Allowed list of this assigned ACL can execute the service. This level of access only applies to services and web service descriptors.
Enforce execute ACL	■ **When top-level service only.** The Integration Server performs ACL checking against the service when it is directly invoked from a client or DSP. For example, suppose a client invokes the OrderParts service on server A. After checking port access, server A checks the Execute ACL assigned to OrderParts to make sure the requesting user is allowed to run the service. It does not check the Execute ACL when other services invoke OrderParts. ■ **Always.** The Integration Server performs ACL checking against the service when it is directly invoked from a client as well as when it is invoked from other services. For example, suppose the OrderParts service is invoked from a browser, as well as by the ProcessOrder and AddToDatabase services. If **Always** is set on OrderParts, the server performs ACL checking on OrderParts three times (once when it is invoked from the browser and twice when it is invoked by ProcessOrder and AddToDatabase). **Note:** This property applies to services only. While you can set an execute ACL for web service descriptors, Integration Server always performs ACL checking when a web service descriptor is called.

Element General Properties

General properties for the element you select in the Package Navigator appear in the Properties view. From the Properties view you can also view and set permissions for the element.

Property	Description
Name	Local name of the element.
Server	Server definition name for the Integration Server on which the element resides.
URL	URL to the Integration Server on which the element resides.

Property	Description
Full name	Fully qualified name of the element.
Package	Package in which the element is located.
REST Resource	Click **Configure...** to open the REST Resource Configuration page and configure REST resources for the selected service. For more information, see "Creating a REST V2 Resource" on page 477

REST Resource Configuration

The REST Resource Configuration page displays the REST resource configured for a selected service.

To open this page, click the service in the Package Navigator of Designer and select **File > Properties > REST Resource Configuration**.

Property	Description
REST URL	The format of the URL that must be followed when clients send REST requests to Designer acting as a REST resource.
Supported Methods	The HTTP methods that the REST URL supports. The following methods are supported: GET, PUT, POST, PATCH, and DELETE.

You can use the following buttons on this page to add, edit, and delete a REST resource configuration:

Use this button...	To...
	Configure a REST resource for the selected service.
	Delete the selected REST resource configuration.
	Edit the selected REST resource configuration.

Document Type Properties

Use the Properties view to view information about IS document types on the Integration Server and to assign permissions, publication properties, and universal names.

To view properties for a document type, double-click the document type in the Package Navigator of Designer.

General Properties for IS Document Types

In the Properties view, under **General**, you can assign an ACL to a document type and view information about the association between IS document type and its source file.

Property	Description
Model type	Specifies the content model for this document type. The content model provides a formal description of the structure and allowed content for a document type which can then be used to validate an instance document. The **Model type** property is display-only. To change the model type for a document type, modify the XML schema definition, and recreate the document type. The contents of an IS document type with a **Model type** property value other than "Unordered" cannot be modified. The **Model type** property can have one of the following values:

Value...	Description...
All	All of the fields in the document type must appear once or not at all, and in any order, in the instance document. The all model type corresponds to an complex type definition that contains an all compositor in the model group.
Choice	One and only one of the fields in the document type can appear in the instance document. The choice model type corresponds to an complex type definition that contains a choice compositor in the model group.
Sequence	Fields in the instance document must appear in the same order in which they are declared in the document type. The sequence model type corresponds to an complex type definition that contains a sequence compositor in the model group.
Simple	A single field that contains simple content and carries an attribute. The document type contains an *@attributeName* field for the attribute value and a **body* field for the simple content.
Unordered	Fields in the instance document can appear in any order and any number of times. A document type has a model type of Unordered if any of the following are true: ■ The document type was created using Developer.

Property	Description
	■ The document type was created using a version of Integration Server prior to version 8.2. ■ The document type was built manually or created from a source file besides an XML schema definition. ■ If the document type was created from an XML schema definition, the Compliance option was not set to “strict” at the time the document type was created. ■ If the document type was created from an XML schema definition that contained repeated or nesting model groups, the any element, or the any attribute and the Compliance option was set to “lax” at the time the document type was created. **Note:** An unordered data structure is sometimes referred to as a “bag” data structure.
Permissions	Click … to assign or view ACL permissions to a document type.
Reuse	Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.
Source URI	Displays the location or URI of the source used to create this document type. If this document type was not based on a source and was instead created from scratch, the **Source URI** property is empty.
Linked to source	Indicates whether the document type reflects the content and structure of the source from which it was created. When set to true, the contents of the document type cannot be edited. When set to false, the document type can be edited but may no longer accurately reflect the content and structure of the source.

Property	Description
Schema domain	Displays the name of the schema domain to which IS schemas generated for this document type belong. This property applies to IS document types created from XML Schema definitions only. **Note:** When Integration Server consumes a WSDL document to create a web service descriptor, Integration Server places any generated IS schemas in a unique schema domain for that web service descriptor. Integration Server uses a combination of alphanumeric characters as the schema domain name.
Registered	Indicates whether Integration Server registered an association between the document type and the complex type definition from which it was created. This is used for derived type support. This property applies to IS document types created from XML Schema definitions only.
Schema type name	Displays the name of the complex type definition with which the IS document type is registered. This is the complex type definition from which the IS document type was created. This property applies to IS document types created from XML Schema definitions only.

webMethods Messaging Properties

In the Properties view, under **webMethods Messaging**, you specify whether a document type can be part of a publish/subscribe solution and the properties of published instances of the document type.

Property	Description
Publishable	Specifies whether the document type can be published and subscribed to.
Connection alias name	Name of the messaging connection alias to use with this publishable document type. The messaging connection alias specifies which provider will receive and route instances of the publishable document type. A value of DEFAULT indicates that default messaging connection alias will be used.
Connection alias type	Displays **Broker** or **Universal Messaging** to indicate which messaging provider is used by the selected alias. This property will be blank if the document type cannot be published to a messaging provider.
Provider definition	Displays the name of the object that corresponds to the publishable document type on the messaging provider. This property displays **Not Publishable** if the document type cannot be published. This property displays **Publishable**

Property	Description
	Locally Only if instances of the document type can be published and subscribed to within this Integration Server only.
Encoding type	Specifies the format used to encode and decode instances of this publishable document type.

Select...	To encode and decode a published document as...
IData	A serialized IData object. An IData object is the universal container that Integration Server uses to receive input and deliver output. An IData object contains an ordered collection of key/value pairs. When a document is encoded as IData, triggers that subscribe to the document type can specify provider filters for the *_properties* header only.
Protocol buffers	A protocol buffer. Protocol buffers is approach to encoding and decoding structured data developed by Google. This is the default. When a document is encoded as a protocol buffer, triggers that subscribe to the document type can specify provider filters for the body of the message only. Note that the body of the message includes the headers as well.

Note:
The **Encoding type** property applies to publishable document types used with a Universal Messaging connection alias only. The **Encoding type** property will be blank for other types of messaging connection aliases. For example, if the publishable document type uses a Broker connection alias or the IS_LOCAL_CONNECTION connection alias, the **Encoding type** property is blank.

Discard	Indicates whether the messaging provider discards instances of this publishable document type after the time specified in the **Time to live** property elapses.

Select...	To...
False	Instructs the messaging provider to never discard instances of this publishable document type.
True	Instructs the messaging provider to discard the document after the period specified in **Time to live** elapses.

Time to live	Specifies how long the messaging provider keeps instances of this publishable document type. If the time to live elapses before a subscriber retrieves the

Property	Description
	document and sends an acknowledgement, the messaging provider discards the document.
Storage type	Specifies whether instances of this document type are stored in memory or on disk.

Select...	To...
Volatile	Volatile documents are stored in memory and provide at most once processing. Volatile documents are never acknowledged and will be lost if the resource on which they reside shuts down.
Guaranteed	Guaranteed documents are saved to disk and provide at least once processing or exactly-once processing. Guaranteed documents will be recovered upon restart if the resource on which they reside shuts down. Resources return acknowledgements for guaranteed documents after successfully storing or processing the document. The acknowledgment allows the sending resource to remove its copy of the document from disk storage.

Property	Description
Validate when published	Specifies whether Integration Server validates published instances of this document type.

Select...	To...
True	Perform validation for published instances of this document type. This is the default.
False	Disable validation for published instances of this document type.

Note: Integration Server ignores the value of the **Validate when published** property if the watt.server.publish.validateOnIS property is set to `always` or `never`.

Universal Name Properties

You can specify a unique public identifier that external protocols (such as SOAP) use to reference a document type on an Integration Server. Every document type on an Integration Server has an explicit universal name in addition to its regular implicit webMethods name. If you omit or delete a document type's explicit universal name, it still retains its implicit universal name.

In the Properties view, under **Universal Name**, you assign a universal name to a document type.

A universal name has two parts: a *namespace name* and a *local name*.

Property	Description
Namespace name	The URI that will be used to qualify the name of this document type. You must specify a valid absolute URI.
Local name	A name that uniquely identifies the document type within the collection encompassed by Namespace name. The name can be composed of any combination of letters, digits, or the period (.), dash (]) and underscore (_) characters. Additionally, it must begin with a letter or the underscore character.

Flat File Dictionary Properties

To view properties for a flat file dictionary, double click the dictionary in the Package Navigator view of Designer. In the Properties view, you can assign list and write access permissions to the flat file dictionary.

To edit the properties for a flat file dictionary, you must have Write access to it and own the lock.

General Properties for a Flat File Dictionary

In the Properties view, under **General**, you can assign an ACL to a flat file dictionary.

Property	Description
Permissions	Click ... to assign or view ACL permissions for a flat file dictionary.
Reuse	Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.

Flat File Element Properties

You use flat file elements to define the structure of a flat file schema and to create the contents of a flat file dictionary. The properties you specify for flat file elements are almost the same. Property descriptions indicate the circumstances in which properties do not apply.

Record Definition Properties

In the Properties view, you can specify properties for a selected record definition in a flat file schema or dictionary.

Property	Description
Ordered	Specifies whether child records must appear in the flat file in the same order in which they appear in the record definition. **Note:** This property applies only to records that are children of this record definition. **Select...** / **To...** **True** — Default. Specify that records in the flat file must appear in the order which they appear in the record definition. If you select **True** and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the records do not appear in the defined order. **False** — Specify that records in the flat file can appear in any order. **Note:** This property does not apply to records in flat file dictionaries.
Mandatory	Specifies whether or not an instance of this record definition is required to exist in the flat file. **Select...** / **To...** **True** — Indicate the record is required. If you select **True** and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the record does not exist in a flat file. **False** — Default. Indicate the record is optional. **Note:** This property does not apply to records in flat file dictionaries.
Max repeat	Maximum number of that instances of this record definition can repeat in the flat file. Set to **Unlimited** if instances of this record definition can repeat any number of times in the flat file. Set to 0 if the record can appear once but cannot repeat. The default is 1, meaning the record can appear once and repeat once.

Property	Description
	If you set the **Max repeat** value to an integer and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the record repeats more than the number of times allowed by the **Max repeat** value. **Note:** This property does not apply to records in flat file dictionaries.
Validator	Specifies the type of validator to use to perform validation for instances of this record definition. Click ... to open the Validators dialog box and select a validator. **Select...** / **To...** **None** — Default. Indicate that no validator is used. **Conditional Validator** — Specify a conditional validator.
Area	The area assigned to this record definition. The **Areas** property for the flat file definition determines the possible values that can be assigned to a record. **Note:** This property does not apply to records in flat file dictionaries.
Position	Integer indicating the position of the of the record in the flat file. Select **Not Used** if you do not want to specify a position for the record. **Note:** This property does not apply to records in flat file dictionaries.
Allow undefined data	Specifies whether an instance of the record definition can contain undefined data and not be considered invalid. A record definition can only allow undefined data if the flat file schema is configured to allow undefined data. (When the Flat File Definition tab is active, in the Properties view, the **Allow undefined data** property is set to True.) **Select...** / **To...** **True** — Default. Allow instances of this record definition to contain undefined data. When a record definition allows undefined data, Integration Server does not generate validation errors when instances of this record reference in a flat file contain undefined data. **False** — Prohibit instances of this record definition from containing undefined data.

Property	Description
	If the *validate* variable of the pub.flatFile.convertToValue service is set to true, the pub.flatFile.convertToValue service will generate errors when undefined data is encountered. **Note:** This property does not apply to records in flat file dictionaries.
Check fields	Specifies whether extra fields in the record instance are considered errors. **Select...** / **To...** **True** — Report errors if, at run-time, the record instances contains more fields than those specified in the record definition. **False** — Default. Allow extra fields at the end of the record instance.
Alternate name	Another name for the record definition. When an IS document type is generated from a flat file schema, the alternate name is used as the name of the document field that corresponds to this record definition.
Local description	This property does not apply to a record definition.
Description	Description of the record definition.

Record Reference Properties

In the Properties view, you an specify properties for a selected record reference in a flat file schema or dictionary.

Property	Description
Ordered	Specifies whether child records must appear in the flat file in the same order in which they appear in the record reference. **Note:** This property applies only to records that are children of this record reference. **Select...** / **To...** **True** — Default. Specify that child records in the flat file must appear in the order which they appear in the record reference. If you select **True** and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the records do not appear in the defined order.

Property	Description	
	False	Specify that child records in the flat file can appear in any order.
Mandatory	Specifies whether or not an instance of this record reference is required to exist in the flat file.	
	Select...	**To...**
	True	Indicate the record is required. If you select **True** and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the record does not exist in a flat file.
	False	Default. Indicate the record is optional.
Max repeat	Maximum number of that instances of this record reference can repeat in the flat file. Set to **Unlimited** if instances of this record reference can repeat any number of times in the flat file. Set to 0 if the record can appear once but cannot repeat. The default is 1, meaning the record can appear once and repeat once. If you set the **Max repeat** value to an integer and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the record repeats more than the number of times allowed by the **Max repeat** value.	
Validator	Specifies the type of validator used to perform validation for instances of this record reference as determined by the referenced record definition.	
	Value...	**Description...**
	None	Indicates that no validator is used.
	Conditional Validator	A conditional validator and the criteria specified by the condition.
Area	The area assigned to this record reference. The **Areas** property for the flat file definition determines the possible values that can be assigned to a record reference.	
Position	Integer indicating the position of the of the record in the flat file. Select **Not Used** if you do not want to specify a position for the record.	
Allow undefined data	Specifies whether an instance of the record reference can contain undefined data and not be considered invalid. A record reference can only allow undefined data if the flat file schema is configured to allow undefined data. (When the Flat File Definition tab is active, in the Properties view, the **Allow undefined data** property is set to True.)	

Property	Description	
	Select...	**To...**
	True	Default. Allow instances of this record reference to contain undefined data. When a record reference allows undefined data, Integration Server does not generate validation errors when instances of this record reference in a flat file contain undefined data.
	False	Prohibit instances of this record reference from containing undefined data. If the *validate* variable of the pub.flatFile.convertToValue service is set to true, the pub.flatFile.convertToValue service will generate errors when undefined data is encountered.
Check fields	Specifies whether extra fields in the record instance are considered errors. This value is determined by the referenced record definition.	
	Value...	**Description...**
	True	Report errors if, at run-time, the record instance contains more fields than those specified in the record definition.
	False	Allow extra fields at the end of the composite instance.
Alternate name	Another name for the record definition. This value is determined by the referenced record definition. When an IS document type is generated from a flat file schema, the alternate name of the record definition is used as the name of the document field that corresponds to this record reference.	
Local description	Description of the record reference. Use a local description to describe the purpose of the record reference in this particular flat file schema.	
Description	Description of the referenced record definition.	

Composite Definition Properties

In the Properties view, you can specify properties for a selected composite definition in a flat file schema or dictionary.

Property	Description	
Mandatory	Specifies whether or not an instance of this composite is required to exist in the flat file.	
	Select...	**To...**

Property	Description	
	True	Indicate the composite is required. If you select **True** and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the composite does not exist in a flat file.
	False	Default. Indicate the composite is optional.
	Note: For a composite definition in a record reference, the **Mandatory** value is determined by the composite definition in the referenced record definition.	
	Note: This property does not apply to composite definitions in flat file dictionaries.	
Extractor	Field number in the record that contains the composite you want to extract. This pulls the field or composite data from the record, or pulls the subfield data from the composite. If you leave this property empty, the composite will not be extracted. Click … to open the Extractors dialog box and specify the extractor. For a composite definition in a record reference, the **Extractor** value is determined by the composite definition in the referenced record definition.	
	Note: The extractor works for a composite only if field delimiters and subfield delimiters have been defined for this flat file schema.	
	Note: This property does not apply to composite definitions in flat file dictionaries.	
Validator	Specifies the type of validator to use to perform validation for instances of this composite definition. Click … to open the Validators dialog box and specify the validator.	
	Select...	**To...**
	None	Default. Indicate that no validator is used.
	Conditional Validator	Specify a conditional validator.

Property	Description
	Note: For a composite definition in a record reference, the **Validator** value is determined by the composite definition in the referenced record definition.
Check fields	Specifies whether extra fields in the composite instance are considered errors. **Select... / To...** **True** — Report errors if, at run-time, the composite instance contains more fields than those specified in the composite definition. **False** — Default. Allow extra fields at the end of the composite instance. **Note:** For a composite definition in a record reference, the **Check fields** value is determined by the composite definition in the referenced record definition.
Local description	For a composite definition in a record reference, specifies a description of the purpose for the composite definition at this particular location in the flat file schema. **Note:** This property does not apply to a composite definition in a record definition. **Note:** This property does not apply to composite definitions in flat file dictionaries.
Alternate name	Another name for the composite definition. When an IS document type is generated from a flat file schema, the alternate name is used as the name of the document field that corresponds to this composite definition. **Note:** For a composite definition in a record reference, the **Alternate name** value is determined by the composite definition in the referenced record definition.
Description	A description for the composite definition. **Note:** For a composite definition in a record reference, the **Description** value is determined by the composite definition in the referenced record definition.
ID Code	IDCode for the composite definition. This information is provided in a SEF file and is used by the WmEDI package.

Property	Description
	Note: For a composite definition in a record reference, the **ID Code** value is determined by the composite definition in the referenced record definition.

Composite Reference Properties

In the Properties view, you can specify properties for a selected composite reference in a flat file schema or dictionary.

Property	Description
Mandatory	Specifies whether or not an instance of this composite is required to exist in the flat file. **Select...** — **To...** **True** — Indicate the composite is required. If you select **True** and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the composite does not exist in a flat file. **False** — Default. Indicate the composite is optional. **Note:** This property does not apply to a composite reference in a record reference. **Note:** For a composite reference in a record reference, the **Mandatory** value is determined by the composite reference in the referenced record definition.
Extractor	Field number in the record that contains the composite you want to extract. This pulls the field or composite data from the record, or pulls the subfield data from the composite. If you leave this property empty, the composite will not be extracted. Click ... to open the Extractors dialog box and select an extractor. **Note:** The extractor works for a composite only if field delimiters and subfield delimiters have been defined for this flat file schema. **Note:** For a composite reference in a record reference, the **Extractor** value is determined by the composite reference in the referenced record definition.

Property	Description
Validator	Specifies the type of validator to use to perform validation for instances of this composite reference as determined by the referenced composite definition.

Value...	Description...
None	Indicates that no validator is used.
Conditional Validator	A conditional validator and the criteria specified by the condition.

Note:
For a composite reference in a record reference, the **Validator** value is determined by the composite reference in the referenced record definition.

Check fields Specifies whether extra fields in the composite instance are considered errors. This value is determined by the referenced composite definition.

Value...	Description...
True	Report errors if, at run-time, the composite instance contains more fields than those specified in the composite definition.
False	Allow extra fields at the end of the composite instance.

Note:
For a composite reference in a record reference, the **Check fields** value is determined by the composite reference in the referenced record definition.

Local description Description of the composite reference. Use a local description to describe the purpose of the referenced composite definition in this particular flat file schema.

Alternate name Another name for the composite definition. This value is determined by the referenced composite definition.

When an IS document type is generated from a flat file schema, the alternate name is used as the name of the document field that corresponds to this composite definition

Note:
For a composite reference in a record reference, the **Alternate name** value is determined by the composite reference in the referenced record definition.

Description Description for the referenced composite definition.

Note:
For a composite reference in a record reference, the **Description** value is determined by the composite reference in the referenced record definition.

Property	Description
ID Code	IDCode for the composite definition. The IDCode is provided in a SEF file and is used by the WmEDI package. **Note:** For a composite reference in a record reference, the **ID Code** value is determined by the composite reference in the referenced record definition.

Field Definition Properties

In the Properties view, specify properties for the selected field definition in a flat files schema or dictionary.

Property	Description
Mandatory	Specifies whether or not an instance of this field definition is required to exist in the flat file. **Select...** / **To...** **True** — Indicate the field is required. If you select **True** and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the field does not exist in a flat file. **False** — Default. Indicate the field is optional. **Note:** This property does not apply to field definitions in flat file dictionaries.
Extractor	Location of the data to extract for this field. Click … to open the Extractors dialog box and specify an extractor. The extractor works for a field only if field delimiters have been defined for this flat file schema. **Select...** / **To...** **Nth Field** — Specify the position of the field to extract. **Note:**Software AG recommends that you use this extractor instead of ID node. **ID Node** — Specify the position of the field to extract. The ID Node extractor is a variation of the Nth Field extractor and is available for backward compatibility for users of the webMethods Module for EDI.

Property	Description
	Fixed Position Specify the start and end positions of the bytes to extract. **Note:** This property does not apply to field definitions in flat file dictionaries.
Validator	Specifies the type of validator to use to perform validation for the field. Click … to open the Validators dialog box and select a validator. **Select...** / **To...** **None** — Default. Indicate that no validator is used. **Length Validator** — Specify the maximum and minimum number of characters this field can contain to be considered valid. **Byte Length Validator** — You specify the maximum and minimum number of bytes this field can contain to be considered valid. **Note:** Use the byte length validator for multi-byte encoded records. **Code List Validator** — Specify a comma–separated list of the allowed values for this field. If the value of the field is not contained in the code list, errors will be generated.
Format service	Enter the fully-qualified name of the service to use to format data from this field. You can click … to navigate to and select a service.
Alternate name	Another name for the field definition. When an IS document type is generated from a flat file schema, the alternate name is used as the name of the String field that corresponds to this field definition.
Description	Description of the field definition. **Note:** This property does not apply to a field definition in a composite reference or a record reference.
ID Code	IDCode for the field definition. The IDCode is provided in a SEF file and is used by the WmEDI package.
Data type	Data type for the field as specified in the SEF file. This information is used by the WmEDI package.
Local description	For a field definition in a composite reference or a record reference, specifies a description for the field definition at this particular location in the flat file schema.

Property	Description
	Note: This property does not apply to field definitions in flat file dictionaries.

Field Reference Properties

In the Properties view, specify properties for the selected field definition in a flat files schema or dictionary.

Property	Description
Mandatory	Specifies whether or not an instance of this field reference is required to exist in the flat file. **Select...** — **To...** **True** — Indicate the field is required. If you select **True** and the *validate* parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the field does not exist in a flat file. **False** — Default. Indicate the field is optional. **Note:** For a field reference in a record reference or composite reference, the **Mandatory** value is determined by the field reference in the referenced record or referenced composite definition.
Extractor	Location of the data to extract for this field. Click ... to open the Extractors dialog box and specify an extractor. The extractor works for a field only if field delimiters have been defined for this flat file schema. **Select...** — **To...** **Nth Field** — Specify the position of the field to extract. **Note:**Software AG recommends that you use this extractor instead of ID node. **ID Node** — Specify the position of the field to extract. The ID Node extractor is a variation of the Nth Field extractor and is available for backward compatibility for users of the webMethods Module for EDI. **Fixed Position** — Specify the start and end positions of the bytes to extract.

Property	Description
	Note: For a field reference in a record reference or composite reference, the **Extractor** value is determined by the field reference in the referenced record or referenced composite definition.
Validator	Specifies the type of validator to use to perform validation for the field as determined by the referenced field definition.

Value...	Description...
None	Default. Indicate that no validator is used.
Length Validator	The maximum and minimum number of characters this field can contain to be considered valid.
Byte Length Validator	The maximum and minimum number of bytes this field can contain to be considered valid. **Note:** Use the byte length validator for multi-byte encoded records.
Code List Validator	A comma–separated list of the allowed values for this field. If the value of the field is not contained in the code list, errors will be generated.

	Note: For a field reference in a record reference or composite reference, the **Validator** value is determined by the field reference in the referenced record or referenced composite definition.
Format service	Enter the fully-qualified name of the service to use to format data from this file as determined by the referenced field definition.
	Note: For a field reference in a record reference or composite reference, the **Format service** value is determined by the field reference in the referenced record or referenced composite definition.
Alternate name	Another name for the field definition as determined by the referenced field definition. When an IS document type is generated from a flat file schema, the alternate name is used as the name of the String field that corresponds to this field definition.
	Note: For a field reference in a record reference or composite reference, the **Alternate name** value is determined by the field reference in the referenced record or referenced composite definition.

Property	Description
Description	Description of the referenced field definition. **Note:** This property does not apply to a field definition in a composite reference or a record reference.
ID Code	IDCode for the field as determined by the referenced field definition. The IDCode is provided in a SEF file and is used by the WmEDI package. **Note:** For a field reference in a record reference or composite reference, the **ID Code** value is determined by the field reference in the referenced record or referenced composite definition.
Data type	Data type for the field as specified in the SEF file. This information is used by the WmEDI package. **Note:** For a field reference in a record reference or composite reference, the **Data type** value is determined by the field reference in the referenced record or referenced composite definition.
Local description	Description of the field reference. Use a local description to describe the purpose of the referenced field definition in this particular flat file schema.

Flat File Schema Properties

To view properties for a flat file schema, double-click the flat file schema in the Package Navigator of Designer. In the Properties view, you can configure the **General**, **Default Record**, and **Settings** properties for the service.

To edit the properties for a flat file schema, you must have Write access to it and own the lock.

General Properties for a Flat File Schema

In the Properties view, under **General**, you can assign an ACL to a flat file schema.

Property	Description
Permissions	Click to assign or view ACL permissions to a flat file schema.
Reuse	Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project.

Property	Description
	When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.

Default Record Properties

In the Properties view, under **Default Record**, you can specify basic information bout the default record for a flat file schema.

Property	Description
Set	Click ... to browse to and select the default record for this flat file schema from a flat file dictionary. This record is used to parse an undefined data record when thepub.flatFile.convertToValues service fails to find a match between the flat file and the flat file schema. If a default record is specified when creating the flat file schema, any record that cannot be recognized will be parsed using this default record. If a default record is not selected, the record will be treated as undefined data. If the **Undefined data** property is set to False and the *validate* variable of the pub.flatFile.convertToValues service is set to true, the pub.flatFile.convertToValues service will generate errors when it encounters undefined data. **Note:** If the flat file you are parsing does not contain record identifiers, you *must* select a default record. By selecting a default record, a CSV (comma separated values) file can be parsed as a special case of record with no record identifier, but with fixed field and record delimiters.
Delete	Click Delete delete the default record for this flat file schema. The actual record definition still exists, but is no longer assigned to this flat file schema.

Settings Properties

In the Properties view, under **Settings**, you can specify whether undefined data is allowed, assign names to particular sections of the flat file schema, and designate a floating record.

Property	Description
Allow undefined data	Specifies whether record definitions and record references in the flat file schema can be configured to allow undefined data. When records allow undefined data, Integration Server does not generate errors when flat files contain undefined data at run time. **Select...** — **To...** **True** — Allow specific records in the flat file schema to be configured to allow undefined data at run time. **False** — Prohibit undefined data at any location in the flat file schema If the *validate* variable of the pub.flatFile.convertToValue service is set to true, the pub.flatFile.convertToValue service will generate errors when undefined data is encountered.
Areas	Click … to create a list of names that can be assigned to records or record references in the flat file schema. **Note:** Areas are used primarily for EDI document parsing.
Floating record	Identifies the record in the flat file schema that will act as the floating record. By designating a floating record, you enable that record to appear in any position within a flat file without causing a parsing validation error. You can specify the record name or the alternate name that is assigned to the record. If you do not use this property, validation errors will occur if the record structure of an inbound document does not match the record structure defined in its flat file schema. For information on avoiding this type of error, see the *Flat File Schema Developer's Guide*.

Schema Definition Properties

The following table identifies the properties for the schema element that functions as the root of the flat file schema.

Property	Description
Ordered	Specifies whether records must appear in the flat file in the same order in which they appear in a flat file schema. **Note:** This property applies only to records that appear at the root of the flat file schema, not records that are child elements of records.

Property	Description	
	Select...	**To...**
	True	Default. Specify that records in the flat file must appear in the order which they appear in the flat file schema. If you select **True** and the validate parameter in the pub.flatFile:convertToValues service is set to true, Integration Server generates errors when the records do not appear in the defined order.
	False	Specify that records in the flat file can appear in any order.

JMS Trigger Properties

Use the following categories of the Properties view to determine the run-time behavior of a JMS trigger.

To view properties for a JMS trigger, double-click the trigger in the Package Navigator of Designer.

General Properties for Non-Transacted JMS Triggers

In the Properties view, under **General**, you specify whether a JMS trigger is enabled, set the transaction type, specify the acknowledgement mode, expiration, and execution user credentials.

Property	Description	
Name	Name of the JMS trigger.	
Enabled	Specifies whether the JMS trigger is enabled or disabled.	
	Select...	**To...**
	True	Enable the JMS trigger
	False	Disable the JMS trigger
	Note: If you disable a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Designer will not retrieve any messages for those web service descriptors.	
Transaction type	Indicates whether or not the JMS trigger receives and processes messages as part of a transaction.	
	Value	**Description**

Property	Description	
	NO TRANSACTION	The JMS trigger does not receive and process message as part of a transaction.
	XA TRANSACTION	The JMS trigger receives and processes messages as part of an XA transaction.
	LOCAL TRANSACTION	The JMS trigger receives and processes messages as part of a local transaction.
	Note: Designer sets the transaction type value based on the transaction type specified for the JMS connection alias associated with the JMS trigger.	
Acknowledgement mode	Indicates how the JMS trigger acknowledges messages it receives to the JMS provider.	
	Select...	**To...**
	AUTO_ACKNOWLEDGE	Automatically acknowledge the message when it is received by the JMS trigger. The Integration Server will acknowledge the message before the trigger completes processing. The JMS provider cannot redeliver the message if Integration Server becomes unavailable before message processing completes.
	CLIENT_ACKNOWLEDGE	Acknowledge or recover the message only after the JMS trigger processes the message completely. This is the default.
	DUPS_OK_ACKNOWLEDGE	Lazily acknowledge the delivery of messages. This may result in the delivery of duplicate messages.
	Note: Designer displays the **Acknowledgement mode** property only if the value of the **Transaction type** property is **NO TRANSACTION**.	
Join expires	Indicates whether the join expires after the time period specified in **Expire after**.	
	Select...	**To...**
	True	Indicates that Integration Server stops waiting for messages from the remaining destination in a join after the time specified in **Expire after** elapses.

Property	Description
	False Indicates that the join does not expire. That is, Integration Server waits indefinitely for messages from the remaining destinations in the join.
	Note: SOAP-JMS triggers do not use joins. Designer does not display the **Join expires** property for SOAP-JMS triggers.
Expire after	Specifies how long Integration Server waits for the remaining documents in the join. The default join time-out is 1 day.
	Note: SOAP-JMS triggers do not use joins. Designer does not display the **Expire after** property for SOAP-JMS triggers.
Reuse	Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.
Execution user	Specifies the name of the user account whose credentials Integration Server uses to execute a service associated with the JMS trigger. You can specify a locally defined user account or a user account defined in a central or external directory.

General Properties for Transacted JMS Triggers

In the Properties view, under **General**, you specify whether a JMS trigger is enabled, set the transaction type, specify the acknowledgement mode, expiration, and execution user credentials.

Property	Description
Name	Name of the JMS trigger.
Enabled	Specifies whether the JMS trigger is enabled or disabled.

Property	Description

Select...	To...
True	Enable the JMS trigger
False	Disable the JMS trigger

Note:
If you disable a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Designer will not retrieve any messages for those web service descriptors.

Transaction type

Indicates whether or not the JMS trigger receives and processes messages as part of a transaction.

Value	Description
NO TRANSACTION	The JMS trigger does not receive and process message as part of a transaction.
XA TRANSACTION	The JMS trigger receives and processes messages as part of an XA transaction.
LOCAL TRANSACTION	The JMS trigger receives and processes messages as part of a local transaction.

Note: Designer sets the transaction type value based on the transaction type specified for the JMS connection alias associated with the JMS trigger.

Execution User

Specifies the name of the user account whose credentials Integration Server uses to execute a service associated with the JMS trigger. You can specify a locally defined user account or a user account defined in a central or external directory.

Reuse

Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project.

When this property is set to public, you can drag the asset to a BPM process or CAF project.

When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project.

All published assets are available for Impact Analysis, whether they are public or private.

Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.

Message Processing Properties

In the Properties view, under **Message processing**, you enable or disable a JMS trigger, set the transaction type, specify the acknowledgement mode, expiration, and execution user credentials.

Property	Description
Processing mode	Specifies how Integration Server processes messages received by the JMS trigger. You can specify serial or concurrent message processing.

Select...	To...
Serial	Specify that Integration Server should process messages received by the trigger one after the other.
Concurrent	Specify that Integration Server should process multiple messages for this trigger at one time.

Max execution threads	Specify the maximum number of messages that Integration Server can process concurrently. Integration Server uses one thread to process each message. The default is 1 server thread.
Max batch messages	Specify the maximum number of messages that the trigger service can receive at one time. If you do not want the trigger to perform batch processing, leave this property set to 1. The default is 1.

Note:
A transacted JMS trigger can be used for batch processing if the JMS connection alias used by the trigger connects to a JMS provider that supports reuse of transacted JMS sessions. If the JMS provider does not support reuse of transacted JMS sessions, set **Max batch processing** to 1. Consult the documentation for your JMS provider to determine whether or not the JMS provider supports the reuse of transacted JMS sessions. Note that webMethods Broker version 8.2 and higher, and Software AG Universal Messaging version 9.5 SP1and higher, and webMethods Nirvana version 7 and higher support the reuse of transacted JMS sessions.

Note: Designer does not display this property for a SOAP-JMS trigger because SOAP-JMS triggers cannot process batches of messages

Connection count	Specifies the number of connections this trigger makes to the JMS provider. Multiple connections can improve trigger throughput, but keep in mind that each connection requires a dedicated Integration Server thread, regardless of the current throughput. The default is 1.

Note:
If you specify a connection count greater than one, the alias associated with this trigger must be configured to create a new connection for each trigger. For more information about JMS connection aliases, refer to *webMethods Integration Server Administrator's Guide*.

Fatal Error Handling Properties

In the Properties view, under **Fatal error handling**, you specify whether the Integration Server should suspend the JMS trigger when an exception occurs during trigger service.

Property	Description
Suspend on Error	Specifies that the Integration Server suspends the JMS trigger when an exception occurs during trigger service execution. This property is available for serial triggers only.

Select...	To...
True	Instruct Integration Server to suspend the trigger when a trigger service ends with a fatal error.
False	Indicate that Integration Server should not suspend the JMS trigger when a trigger service ends with a fatal error This is the default.

Transient Error Handling with a Non-Transacted JMS Trigger

When building a trigger, you can specify what action Integration Server takes when the trigger service fails because of a transient error caused by a run-time exception. That is, you can specify whether or not Integration Server should retry the trigger.

In the Properties view, under **Transient error handling**, you specify whether or not Integration Server should retry the trigger.

Property	Description
Max retry attempts	Specifies the maximum number of times Integration Server should re-execute the trigger service when the trigger service ends because of a transient error that causes an ISRuntimeException. The default is 0 attempts (indicating the trigger service does not retry).
Retry interval	Specifies the length of time Integration Server waits between retry attempts. The default is 10 seconds.
On retry failure	Indicates how Integration Server handles a retry failure for a JMS trigger. A retry failure occurs when Integration Server reaches the maximum number of retry attempts and the trigger service still fails because of an ISRuntimeException. This property also determines how Integration Server handles a transient error that occurs during trigger preprocessing.

Select...	To...

Property	Description	
	Throw exception	Indicate that Integration Server throws a service exception when the last allowed retry attempt ends because of an ISRuntimeException. This is the default.
	Suspend and retry later	Indicate that Integration Server suspends the trigger when the last allowed retry attempt ends because of an ISRuntimeException. Integration Server retries the trigger service at a later time when the resources needed by the trigger service become available. When **On Retry failure** is set to **Suspend and retry later**, a transient error that occurs during trigger preprocessing causes Integration Server to suspend the trigger and resume it when the resources, specifically the document history database, are available.
On transaction rollback	Indicates how Integration Server handles a transient error that occurs during service execution, resulting in the entire transaction being rolled back.	
	Select...	**To...**
	Recover only	Indicate Integration Server recovers the message back to the JMS provider. Integration Server receives the message again almost immediately. This is the default.
	Suspend and recover	Indicate that Integration Server suspends the JMS trigger and then recovers the message back to the JMS provider. Integration Server executes the trigger service at a later time when the resources needed by the trigger service become available.
Resource monitoring service	Specifies the service that Integration Server executes to determine whether the resources needed by the JMS trigger are available and if the trigger can be resumed. Integration Server schedules a system task to execute the resource monitoring service when one of the following occurs: ■ The trigger service ends because of a retry failure and the **On retry failure** property is set to **Suspend and retry later**. ■ The trigger service is part of a transacted JMS trigger and the **On transaction rollback** property is set to **Suspend and recover**. ■ The document resolver service used for exactly-once processing ends because of a run-time exception and the watt.server.trigger.preprocess.suspendAndRetryOnError is set to true.	

Property	Description
	Note: A resource monitoring service must use the pub.trigger:resourceMonitoringSpec as the service signature.

Transient Error Handling with a Transacted JMS Trigger

When building a trigger, you can specify what action Integration Server takes when the trigger service fails because of a transient error caused by a run-time exception. That is, you can specify whether or not Integration Server should retry the trigger.

In the Properties view, under **Transient error handling**, you specify whether or not Integration Server should retry the trigger.

Property	Description
On transaction rollback	Indicates how Integration Server handles a transient error that occurs during service execution, resulting in the entire transaction being rolled back. **Select...** / **To...** **Recover only** — Indicate Integration Server recovers the message back to the JMS provider. Integration Server receives the message again almost immediately. This is the default. **Suspend and recover** — Indicate that Integration Server suspends the JMS trigger and then recovers the message back to the JMS provider. Integration Server executes the trigger service at a later time when the resources needed by the trigger service become available.
Resource monitoring service	Specifies the service that Integration Server executes to determine whether the resources needed by the JMS trigger are available and if the trigger can be resumed. Integration Server schedules a system task to execute the resource monitoring service when one of the following occurs: ■ The trigger service ends because of a retry failure and the **On retry failure** property is set to **Suspend and retry later**. ■ The trigger service is part of a transacted JMS trigger and the **On transaction rollback** property is set to **Suspend and recover**. ■ The document resolver service used for exactly-once processing ends because of a run-time exception and the watt.server.trigger.preprocess.suspendAndRetryOnError is set to true. **Note:**

Property	Description
	A resource monitoring service must use the pub.trigger:resourceMonitoringSpec as the service signature.

Exactly Once Processing Properties

You can configure exactly-once processing for a JMS trigger. Exactly-once processing ensures that a trigger processes a persistent message once and only once.

Integration Server provides exactly-once processing by determining whether a message is a copy of one previously processed by the trigger. Designer provides three duplicate detection methods: redelivery count, document history, and a document resolver service.

In the Properties view, under **Exactly once**, you configure exactly-once processing for a JMS trigger.

Property	Description
Detect duplicate	Enables exactly-once processing for the JMS trigger and instructs the server to check a message's redelivery count to determine whether the trigger has received the message before.

Select...	To...
True	Specifies that exactly-once processing is provided for messages received by this trigger and instructs the server to check a document's redelivery count to determine whether the trigger received the document previously. The redelivery count indicates the number of times the routing resource has redelivered a document to the trigger.
False	Specifies that exactly-once processing is not provided for messages received by this trigger.

Note: Integration Server provides exactly-once processing for guaranteed messages only.

Property	Description
Use history	Indicates whether a document history database will be maintained and used to determine whether a message is a duplicate.

Select...	To...
True	Indicates that Integration Server maintains a history of messages processed by the trigger. When the trigger receives a message, the Integration Server compares the message's universally unique identifier (UUID) to the UUIDs of messages processed by the trigger. If there is a match, the Integration Server either determines the second message is a duplicate and discards it or,

Property	Description
	if the first message has not finished processing, marks the second message's status as **In Doubt**.
	False Indicates Integration Server does not maintain a document history database. The Integration Server will not use document history to determine whether a message is a duplicate of one already processed by the trigger.
	Note: To perform duplicate detection using a document history database, the audit subsystem must be stored in a relational database and the Integration Server Administrator must define a JDBC connection pool for the Integration Server to use to connect to the document history database.
History time to live	Specifies the length of time the document history database maintains an entry for a message processed by the JMS trigger. During this time period, the Integration Server discards any messages with the same universally unique identifier (UUID) as an existing document history entry for the trigger. When a document history entry expires, the Integration Server removes it from the document history database. If the trigger subsequently receives a message with same UUID as the expired and removed entry, the server considers the copy to be new because the entry for the previous message has been removed from the database.
Document resolver service	Specifies the service that you created to determine whether message's status is **New**, **Duplicate**, or **In Doubt**. Click … to select a service from a list. The document resolver service must use the pub.publish:documentResolverSpec to define the service signature.

webMethods Broker Properties

In the Properties view, under **webMethods Broker**, you can configure properties specific to webMethods Broker when the Broker is used as the JMS provider.

Property	Description
Max prefetch size	Specifies the maximum number of messages that the webMethods Broker API for JMS will retrieve and cache per request for this trigger. Using pre-fetch cache can speed up the retrieval of messages from webMethods Broker. Because messages will be placed in Integration Server memory, you may want to decrease this setting if this trigger receives very large messages. Otherwise, memory issues can occur. This setting applies only if this trigger connects to webMethods Broker. The default is 10 messages.

Property	Description
	If you specify a value of -1 for this setting, Integration Server uses the value specified on the watt.server.jms.trigger.maxPrefetchSize parameter for this setting.

Link Properties

Use the Properties view to apply conditions to the link you have drawn between two variables or specify which element of an array you want to link to or from.

To view properties for a link, double-click the link in the Pipeline editor.

General Properties for Links

Property	Description	
Evaluate copy condition	Specifies whether Integration Server evaluates a condition applied to a link before executing the link.	
	Select...	**To...**
	True	Instruct Integration Server to execute the link only if the condition specified in **Copy Condition** evaluates to true.
	False	Instruct Integration Server to execute the link without evaluating it against a copy condition.
Copy condition	Specifies the expression that must evaluate to true before the Integration Server will execute the link between fields. The server evaluates the condition at run time only if the **Evaluate copy condition** property is set to **True**. Click … to specify a condition. Use the syntax provided by webMethods to write the condition. For details on the syntax, see "Conditional Expressions" on page 1113.	
Indices	To specify which element in an array variable you want to link to or from, click …. See ct one of the following from the Link Indices page:	
	Select...	**To...**
	Source	Display the variables for which you need to specify an array index. You only need to specify indexes for Source variables if the source variable (the variable you are linking from) is an array or is a child of an array.
		Index

Property	Description	
		Specifies the array index for the element whose value you want to link to the target variable
	Destination	Displays the variables for which you need to specify an array index. You only need to specify indexes for Destination variables if the target variable is an array or is the child of an array.
		Index Specifies the array index for the element to which you want to link the source variable.

OData Service Properties

To view the properties for an OData service and its elements, properties, and associations, double-click the service in the Package Navigator of Designer. You can configure the properties for the service in the Properties view.

To edit the properties for a service, you must have Write access to it and own the lock.

General Properties for OData Services

Property	Description	
Permissions	Click … to assign or view ACL permissions to an OData service.	
Alias	Specifies an alternate name for the namespace name of the OData service.	
Namespace	Displays the namespace name, which is the fully qualified name of the OData service on the Integration Server.	
Use custom filter	Indicates whether or not to use custom filters instead of the built-in filters that Integration Server provides while using the $filter system query option.	
	Select...	**To...**
	True	Use your custom filters in the $filter system query option. You can then specify custom filter queries as the value for the *$filter* parameter of the _retrieve and _update OData service implementations.
	False	Use the built-in filters that Integration Server provides in the $filter system query option. This is the default.

OData Element Properties

In the Properties view, under **General**, you can specify the name for the **Entity Type** and **Complex Type** OData elements.

Entity Type Properties

Property	Description
Name	Specifies the name of the OData entity type. Name of an entity type must be unique among all the entity types in the OData service.

Complex Type Properties

Property	Description
Name	Specifies the name of the OData complex type. Name of a complex type element must be unique across all complex types in the OData service.

External Entity Type Properties

Property	Description
Name	Specifies the name of the OData external entity type.
Connection Alias	Specifies the connection used with the external source type.
Source Reference Name	Specifies the name of the source external entity.

Note:
You cannot edit the properties of an **External Entity Type**.

Simple Property Properties

In the Properties view, you can specify the general properties for a selected Simple property in an OData service.

General Properties for Simple Property

Property	Description
Name	Specifies the name of the Simple property. The name of the property must be unique within the set of Simple properties for the entity type or complex type.
Key	Indicates whether or not the OData element is a key. This property is available only for the Simple property of OData entity types. Each OData entity type must have a **Key** property that uniquely identifies the entity type within the OData service at run time. **Select... / To...** **True** — Indicate that the property is a key. When selected, the icon representing the OData Simple property in the OData service editor changes to . **False** — Indicate that the property is not a key property. This is the default.
Nullable	Indicates whether or not the property can have a null value. **Note:** If you selected **True** for the **Key** property, the property cannot have a null value. **Select... / To...** **True** — Indicate that property can have a null value. This is the default. **False** — Indicate that the property cannot have a null value.
Type	Specifies the OData-specific primitive data type of the property.

Facets Properties for Simple Property

In the Properties view, you can specify the constraining facets to be applied to the specified primitive type of the selected Simple property in an OData service.

Property	Description
Default	Determines the default value of the property. Enter the default value or select **NULL** if the default value is null. **Note:** If the OData element is a key, that is, if you selected `True` for the **Key** property, the **Default** property cannot have a null value.
Fixed Length	Specifies whether the length of the value of the property must be fixed or whether it can vary.

Property	Description	
	Select...	**To...**
	True	Indicate that the length of the value of the property consists of a fixed number of bytes.
	False	Indicate that the value of the property can be of varying length.
Max Length	Specifies the maximum length of the value of the property. Enter a positive integer if you want to restrict the value to a specific length. Select **Max** if the value can be of any length.	
Unicode	Specifies whether or not the value of the property is encoded using Unicode (UTF-8) or ASCII. .	
	Select...	**To...**
	True	Indicate that value of the property encoded using Unicode (UTF-8). This is the default.
	False	Indicate that the value of the property is encoded using ASCII.
Collation	Specifies a sorting sequence that can be used for comparison and ordering operations on values of the property.	

Complex Property Properties

In the Properties view, you can specify the general properties for a selected Complex property in an OData service.

Property	Description
Name	Specifies the name of the Complex property of the OData service. The name of the property must be unique within the set of Complex properties for the entity type or complex type.
Type	Specifies the primitive data type of the property.

Association Properties

In the Properties view, you can specify the properties for the OData associations.

General Properties for Association

Property	Description
Name	Specifies the name of the OData Association. The value of this property is derived from the two entity types that are part of this entity association and the multiplicity in this association.

OData Association End Properties

In the Properties view, you can specify the properties for the OData association ends.

Property	Description
Entity Type	Specifies the Entity Type on the specific association end.
Multiplicity	Specifies the number of entity types that can be at the specific end of the association.

Select...	To...
*	Indicate that zero, one, or more entity type instances exist at the association end.
0..1	Indicate that zero or one entity type instance exist at the association end.
1	Indicate that exactly one entity type instance exists at the association end.

Role	Specifies the name of the role played by the entity type at an association end.

OData Association Navigation Properties

When an association is defined between entity types, Designercreates and displays the OData navigation elements under the corresponding entity types. In case of single or unidirectional association, a navigation element is added only to one association end. In case of bidirectional association, navigation elements are added to both the association ends. You can view the properties for the OData navigation elements in the Properties view.

Property	Description
Name	Specifies the name of the navigation property. The value is automatically derived when an association is formed between two entity types.
Relationship	Specifies the association related to this navigation property.
From Role	Specifies the name of the role played by this entity type in the association.
To Role	Specifies the name of the role played by this entity type in the association.

REST V2 Resource Properties

To view the properties of a REST V2 resource, double click the resource in the package navigator view.

The properties view displays the following information for the selected REST V2 resource operation:

Property	Description
Permissions	Click ... to assign or view ACL permissions to the REST V2 resource.
Reuse	Indicates if the resource must be reused by other assets belonging to the Integration Server (Private) or by external entities (Public).
Source URI	Displays the location of the source used to create this resource. The value for this property appears *only* for a REST V2 resource associated with a descriptor that is generated from a Swagger document.

REST API Descriptor Properties

To view properties for a REST API descriptor, double-click the REST API descriptor in Package Navigator view of Designer.

General Properties for REST API Descriptors

In the Properties view, under **General**, you can view and assign properties to a REST API descriptor.

Property	Description
Name	Displays the name of the REST API descriptor.
Permissions	Click ... to assign or view ACL permissions to the REST API descriptor.
Reuse	Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. **Note:** A REST API descriptor cannot be published to CentraSite.

Property	Description
Source URI	The URI of the source Swagger document used to generate the REST API descriptor. This property is not used for a REST API descriptors that is not generated from a Swagger document.
Source URL	The URL to access the source Swagger document used to generate the REST API descriptor. This property is not used for a REST API descriptors that is not generated from a Swagger document.
Match payload with Swagger	Specifies whether Integration Server matches the payload of REST API requests with the Swagger definition or the service input signature. When the property is set to `true` (the default), Integration Server matches the request payload with the Swagger definition. When the property is set to `false`, Integration Server matches the request payload with the service input signature. This property is introduced for PIE-71082 in IS_10.1_Core_Fix24.

REST Resource Properties

When you select a resource on the REST Resources tab, the Properties view displays path and suffix information for the selected resource.

Property	Description
Path	The path for the REST resource. By default, each REST resource in a REST API descriptor derives its path from the namespace of the REST resource. **Note:** The value of this property *cannot* be edited for REST API descriptors containing REST V2 resources. For a REST resource created using the legacy approach, you can override the default path with a custom value. For example, you could use `/customers/premium` or `/myPath`. Change the path of the REST resource to be the path of your choosing. If you do not include "/" as the first character in the **Path** property, Integration Server adds it in the Swagger document. Ensure that Integration Server can resolve the path that you specify. Integration Server must be able to invoke the path.
Suffix	The suffix for the REST resource. **Note:**

Property	Description
	You can add a suffix *only* if the descriptor contains REST resources created using the legacy approach. This property is not used for descriptors containing REST V2 resources. By default, there is no suffix for a REST resource in a REST API descriptor. However, if you want users who invoke the REST resource to include query parameters, you can specify that information in the suffix. Integration Server appends the suffix to the **Path** property value. For example, if you want the exposed path in the Swagger document to be `/customers /{id}`, specify `/customers` as the **Path** property value and `/{id}` as the **Suffix** property value. Ensure that Integration Server can resolve the resource path when it includes the suffix that you specify. Integration Server must be able to invoke the resource path.

Operation Properties

When you select an operation in a REST resource on the REST Resources tab, the Properties view displays the operation name and description.

Property	Description
Operation name	Fully qualified name of the selected operation.
Operation description	Description of the selected operation.

REST Definition Properties

The REST Definitions tab contains information about multiple REST definitions contained in a Swagger Document. Each REST Definition is a root-level document on the REST Definitions tab and can have multiple parameters.

Property	Description
Name	Fully qualified name of the REST definition.
Description	This property is not used.
Namespace Name	If a service signature for a service in a REST resource includes a reference to an IS document type, Integration Server creates a REST definition that corresponds to the IS document type. For these REST definitions, the **Namespace Name** is the fully qualified name of the referenced IS document type.

Property	Description
	The **Namespace Name** property applies to a REST definition created for an IS document type only.

REST Definition Parameter Properties

When you select a parameter on the REST Definitions tab, the Properties view displays information about the parameter, such as the name, description, and whether or not it is required.

Property	Description
Name	Parameter name. The parameter name corresponds to the name of a variable in the signature of the service used as an operation in the REST API descriptor.
Description	Parameter description. Integration Server obtains the description from the **Comments** property value for the variable in the signature of the service used an operation in the REST API descriptor.
Reference definition	If the parameter is a reference to a REST definition, displays the name of the referenced REST definition.
Required	Whether or not the parameter is required. A value of **True** indicates the parameter is squired. A value of **False** indicates the parameter is optional.

Schema Properties

In the Properties view, you can view and set the properties for an IS schema. To view the properties for an IS schema, double-click the schema in Package Navigator view.

To edit the properties for a specification, you must have Write access to it and own the lock.

General Properties for IS Schemas

In the Properties view, under **General**, you can assign list or write ACL privileges to an IS schema or indicate how it can be used in CentraSite.

Property	Description
Permissions	Click … to assign or view ACL permissions to a schema.
Schema domain	Displays the name of the schema domain to which the IS schema belongs. This property applies to schemas created from XML Schema definitions only.

Property	Description
	Note: When Integration Server consumes a WSDL document to create a web service descriptor, Integration Server places any generated IS schemas in a unique schema domain for that web service descriptor. Integration Server uses a combination of alphanumeric characters as the schema domain name.
Reuse	Specifies whether this schema can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.
Source URI	Displays the location or URI of the source used to create this schema.
Linked to source	Indicates whether the schema reflects the content and structure of the source from which it was created. When set to true, the contents of the schema, specifically simple type definitions, cannot be edited. When set to false, the simple type definitions in the schema can be edited but may no longer accurately reflect the simple type definitions from the source.

Schema Component Properties

In the schema editor, you can view the contents of an IS schema and details about each component in the schema. When you select a component in the Schema Browser (the left side of the schema editor), Designer displays detailed information in the Component Details (the right side of the schema editor). The information contained in Component Details varies with the selected component.

All Content Model

An all content model specifies that child elements can appear once, or not at all, and in any order in the instance document. This symbol corresponds to the `<all>` compositor in an XML Schema definition.

This field...	Specifies...
Min Occurs	The minimum number of occurrences of the content model for an element in the instance document. The value of **Min Occurs** is equal to the value of the `minOccurs` attribute in the `<all>` content model
Max Occurs	The maximum number of occurrences of the content model for an element in the instance document. The value of **Max Occurs** is equal to the value of the `maxOccurs` attribute in the `<all>` content model.
Summary of Children	The name and occurrence constraints for the child elements in the content model. ■ **Name**. The name of the child element. ■ **Min, Max**. The minimum and maximum occurrence constraints for the child element. The **Min** and **Max** values correspond to the `minOccurs` and `maxOccurs` attributes (respectively) in the local element declaration.

Any Attribute Declaration

An any attribute declaration is a wildcard declaration used as a placeholder for undeclared attributes in an instance document. This symbol corresponds to the `<anyAttribute>` declaration in an XML Schema. The attribute that corresponds to the wildcard declaration in the instance document is called the matching attribute.

Because an `<anyAttribute>` declaration does not specify an attribute name, the schema uses 'Any' as the name of the attribute.

This field...	Specifies...
Process Contents	The validation constraints placed on the matching attribute in the instance document. The value of **Process Contents** equals the value of the `processContents` attribute in the `<anyAttribute>` declaration. This field can have one of the following values: ■ **strict** specifies that Integration Server must validate the matching attribute against a global declaration in a schema belonging to one of the allowed namespaces. If Integration Server cannot find one of the XML components, it generates a validation error. The **Process Contents** value is strict when the `processContents` attribute is absent or is set to "strict" in the `<anyAttribute>` declaration. This is the default value. ■ **skip** specifies that the matching attribute must be well-formed XML. Integration Server does not need to make sure the matching element or attribute is schema-valid. The **Process Contents** value is skip when `processContents="skip"` in the `<anyAttribute>` declaration.

This field...	Specifies...
	■ **lax** specifies that when Integration Server encounters the matching attribute in the instance document, it should (if possible) validate the matching attribute against the corresponding global declaration in a schema from an allowed namespace. However, if Integration Server cannot find the schema in the namespace, no error occurs. The **Process Contents** value is lax when `processContents="lax"` in the `<anyAttribute>` declaration.
Qualifier	Whether the matching attribute can or cannot be from one of the namespaces listed in the URIs field. The namespace attribute value in the `<anyAttribute>` declaration determines the value of **Qualifier**. See the following.

Namespace value	Qualifier value	Description
##any	any	Specifies that the matching attribute can be from any namespace.
##local	specific	Specifies that the matching attribute must belong to a namespace
##targetNamespace	specific	Specifies that the matching attribute must be from the namespace listed in the URIs field.
##other	not	Specifies that the matching attribute must be from a namespace other than the namespaces listed in the URIs field.
"*URI1 URI2*"	specific	Specifies that the matching attribute must be from one of the namespaces listed in the URIs field.

URIs The namespaces to which the matching attribute can or cannot belong. The namespace attribute value in the `<anyAttribute>` declaration determines the value of URIs. See the following.

Namespace value	URIs
any	blank. If the namespace attribute does not appear in the attribute declaration, ##any is used.
##local	"unqualified"
##other	*target namespace* and "unqualified"
##targetNamespace	*target namespace*

This field...	Specifies...
"*URI1 URI2*"	Specified by the namespace attribute in the `<anyAttribute>` declaration.

Any Element Declaration

An any element declaration in XML Schema is a wildcard declaration used as a placeholder for one or more undeclared elements in an instance document. The element that corresponds to the wildcard declaration in the instance document is called the matching element. In a DTD, an element declared to be of type ANY can contain any well-formed XML. This symbol corresponds to an `<any>` element declaration in an XML Schema and an element declared to be of type ANY in a DTD.

Because an `<any>` element declaration does not have a name, the schema uses 'Any' as the name of the element.

This field...	Specifies...
Min Occurs	The minimum number of occurrences for the matching element in the instance document.
Max Occurs	The maximum number of occurrences for the matching element in the instance document.
Process Contents	The validation constraints placed on the matching element in the instance document. The value of Process Contents equals the value of the `processContents` attribute in the `<any>` declaration. This field can have one of the following values: ■ **strict** specifies that Integration Server must validate the matching element against a global declaration in a schema belonging to one of the allowed namespaces. If Integration Server cannot find one of the XML components, it generates a validation error. The **Process Contents** value is strict when the `processContents` attribute is absent or is set to "strict" in the `<any>` declaration. This is the default value. ■ **skip** specifies that the matching element must be well-formed XML. Integration Server does not need to make sure the matching element is schema-valid. The **Process Contents** value equals skip when `processContents="skip"` in the `<any>` declaration. ■ **lax** specifies that when Integration Server encounters the matching element in the instance document, it should (if possible) validate the matching element against the corresponding global declaration in a schema from an allowed namespace. However, if Integration Server cannot find the schema in the namespace, no error occurs. The **Process**

This field...	Specifies...
	Contents value equals lax when `processContents="lax"` in the `<any>` declaration.
Qualifier	Whether the matching element can or cannot be from one of the namespaces listed in the URIs field. The namespace attribute value in the `<any>` declaration determines the value of **Qualifier**. See the following.

Namespace value	Qualifier value	Description
##any	any	Specifies that the matching element can be from any namespace.
##local	specific	Specifies that the matching element must belong to a namespace.
##other	not	Specifies that the matching element must be from a namespace other than the namespaces listed in the URIs field.
##targetNamespace	specific	Specifies that the matching element must be from the namespace listed in the URIs field
"*URI1 URI2*"	specific	Specifies that the matching element must be from one of the namespaces listed in the URIs field.

URIs The namespaces to which the matching element can or cannot belong. The namespace attribute value in the `<any>` declaration determines the value of URIs. See the following.

Namespace value	URIs
any	blank. If the namespace attribute does not appear in the `<any>` declaration, ##any is used.
##local	"unqualified"
##other	*target namespace* and "unqualified"
##targetNamespace	*target namespace*

This field...	Specifies...
"*URI1 URI2*"	Specified by the namespace attribute in the `<any>` declaration.

Attribute Declaration

An attribute declaration associates an attribute name with a simple type definition. This symbol corresponds to the XML Schema `<attribute>` declaration or the attribute in a DTD ATTLIST declaration.

An attribute declaration can specify a default value, a fixed value, and whether the appearance of the attribute in the instance document is required. Like element declarations, attribute declarations can be global or local.

This field...	Specifies...
Name	The local name and target namespace of the attribute declaration. The **Name** value is equal to the expanded value (prefix plus local name) of the `name` attribute in the attribute declaration.
Default	The default value for the attribute in an instance document. The **Default** value is equal to the value of the default attribute in the attribute declaration. During data validation, Integration Server supplies the instance document with an attribute whose value equals that of **Default** if: ■ The element to which the attribute is assigned appears in the instance document, and ■ The attribute itself does not appear. If the element to which the attribute declaration is assigned does not appear in the instance document, Integration Server does not augment the instance document. **Note:** During data validation, Integration Server applies the **Default** value for an attribute declaration differently than the **Default** value for an element declaration. For attributes, Integration Server supplies the instance document with an attribute with the Default value when the attribute does not appear in the instance document. For elements, Integration Server supplies the instance document with an element equal to the **Default** value only if the element appears in the instance document, but has no content.

This field...	Specifies...
Fixed Value	The fixed value for the attribute. The **Fixed Value** is equal to the value of the fixed attribute in the attribute declaration. If this attribute appears in an instance document, the attribute value must equal the **Fixed Value**. During data validation, Integration Server supplies the instance document with an attribute whose value equals **Fixed Value** if: ■ The element to which the attribute is assigned appears in the instance document, and ■ The attribute itself does not appear in the instance document.
Is Required	Whether or not the attribute must appear in an instance document. **Is Required** is determined by the value of the use attribute in the attribute declaration. This field only appears when you select a local attribute declaration or an attribute reference. The **Is Required** field can have one of the following values: ■ **True** specifies that the attribute is required and must appear in the instance document. When the value of **Is Required** is True, the attribute declaration contains `use="required"`. ■ **False** specifies that the appearance of the attribute in the instance document is optional; that is, the attribute is not required. When the value of **Is Required** is False, the attribute declaration or `use="optional"` or the use attribute is absent from the declaration. If an attribute declaration in an XML Schema contains `use="prohibited"`, the attribute declaration does not appear in the schema browser. Integration Server does not retain attribute declarations whose use is prohibited.
Simple Type	The name and namespace of the simple type definition associated with the attribute. See "Simple Type Definition" on page 1021.

Attribute Reference

An attribute reference is a reference from a complex type definition to a globally declared attribute. This symbol corresponds to the `ref="globalAttributeName"` attribute in an attribute declaration. DTDs do not have attribute references. Consequently, attribute references do not appear in schemas generated from DTDs.

This field...	Specifies...
Name	The local name and target namespace of the attribute declaration for this attribute reference. The **Name** value is equal to the expanded value (prefix plus local name) of the `name` attribute in the attribute declaration.
Default	The default value for the referenced attribute in an instance document. The **Default** value is equal to the value of the default attribute in the attribute declaration. During data validation, Integration Server supplies the instance document with an attribute whose value equals that of **Default** if: ■ The element to which the attribute is assigned appears in the instance document, and ■ The attribute itself does not appear. If the element to which the attribute declaration is assigned does not appear in the instance document, Integration Server does not augment the instance document. **Note:** During data validation, Integration Server applies the **Default** value for an attribute declaration differently than the **Default** value for an element declaration. For attributes, Integration Server supplies the instance document with an attribute with the Default value when the attribute does not appear in the instance document. For elements, Integration Server supplies the instance document with an element equal to the **Default** value only if the element appears in the instance document, but has no content.
Fixed Value	The fixed value for the referenced attribute. The **Fixed Value** is equal to the value of the fixed attribute in the attribute declaration. If this attribute appears in an instance document, the attribute value must equal the **Fixed Value**. During data validation, Integration Server supplies the instance document with an attribute whose value equals **Fixed Value** if: ■ The element to which the attribute is assigned appears in the instance document, and ■ The attribute itself does not appear in the instance document.
Is Required	Whether or not the referenced attribute must appear in an instance document. **Is Required** is determined by the value of the use attribute in the attribute declaration. This field only appears when you select a local attribute declaration or an attribute reference. The **Is Required** field can have one of the following values:

This field...	Specifies...
	■ **True** specifies that the attribute is required and must appear in the instance document. When the value of **Is Required** is True, the attribute declaration contains `use="required"`. ■ **False** specifies that the appearance of the attribute in the instance document is optional; that is, the attribute is not required. When the value of **Is Required** is False, the attribute declaration or `use="optional"` or the use attribute is absent from the declaration. If an attribute declaration in an XML Schema contains `use="prohibited"`, the attribute declaration does not appear in the schema browser. Integration Server does not retain attribute declarations whose use is prohibited.
Simple Type	The name and namespace of the simple type definition associated with the referenced attribute. See "Simple Type Definition" on page 1021.

Choice Content Model

A choice content model specifies that only one of the child elements in the content model can appear in the instance document. This symbol corresponds to the `<choice>` compositor in an XML Schema definition or a choice list in a DTD element type declaration.

If one of the child elements does not appear or more than one child element appears, the instance document is not schema-valid. (An exception to this is when the `minOccurs` attribute for the `<choice>` element is set to 0. If `minOccurs=0`, Integration Server does not generate a validation error if no child element appears.)

This field...	Specifies...
Min Occurs	The minimum number of occurrences of the content model for an element in the instance document. The value of **Min Occurs** is equal to the value of the `minOccurs` attribute in the `<choice>` content model
Max Occurs	The maximum number of occurrences of the content model for an element in the instance document. The value of **Max Occurs** is equal to the value of the `maxOccurs` attribute in the `<choice>` content model.
Summary of Children	The name and occurrence constraints for the child elements in the content model. ■ **Name**. The name of the child element. ■ **Min, Max**. The minimum and maximum occurrence constraints for the child element. The **Min** and **Max** values correspond to the

This field...	Specifies...
	`minOccurs` and `maxOccurs` attributes (respectively) in the local element declaration.

Complex Type Definition

A complex type definition defines the structure and content for elements of complex type. (Elements of complex type can contain child elements and carry attributes.) This symbol corresponds to the `<complexType>` element in an XML Schema definition.

If the complex type definition is unnamed (an anonymous type), the Schema Browser displays 'Anonymous' as the name of the complex type definition.

This field...	Specifies...
Name	The local name and target namespace of the complex type. The **Name** value is equal to the expanded value (prefix plus local name) of the name attribute in the type definition. If the Schema Browser displays 'Anonymous' as the name of the simple type, the complex type is an anonymous (unnamed) type defined in an element declaration.
Is Abstract	Whether the complex type definition is abstract. The value of **Is Abstract** corresponds to the `abstract` attribute in the complex type definition. The **Is Abstract** field can have one of the following values: ■ **True** indicates the complex type is abstract. When an element corresponds to an abstract complex type definition, the element in the instance document must contain `xsi:type` to refer to a complex type derived from the abstract type. When the value of **Is Abstract** is True, the complex type definition contains `abstract="true"`. ■ **False** indicates the complex type definition is not abstract. When the value of **Is Abstract** is False, the abstract attribute is absent from the complex type definition or `abstract="false"`.

Note:
If the complex type was created from a simple type, then the Schema Browser also displays the fields for the simple type. For details, see "Simple Type Definition" on page 1021.

Element Declaration

An element declaration associates an element name with a type definition. This symbol corresponds to the `<element>` declaration in an XML Schema and the ELEMENT declaration in a DTD.

An element declaration can contain attributes to specify a default value, a fixed value, and whether the element is abstract or nillable. If an element declaration is part of a content specification, the element declaration can contain attributes to specify occurrence constraints.

This field...	Specifies...
Name	The local name of the element followed by the namespace to which the element belongs. The **Name** value is equal to the expanded value (prefix plus local name) of the name attribute in the element declaration.
Default	The default value for the element. The **Default** value is equal to the value of the default attribute in the element declaration. During data validation, if the element appears in an instance document but contains no content, Integration Server supplies the element with the **Default** value. If the element does not appear in the instance document, Integration Server does not augment the instance document.
Fixed Value	The fixed value for the element. The **Fixed Value** is equal to the value of the `fixed` attribute in the element declaration. When Integration Server validates an instance document against the schema, if the element appears, its value must be equal to the **Fixed Value**. During data validation, if the element appears in an instance document but contains no content, Integration Server supplies the element with the **Fixed Value**. If the element does not appear in the instance document, Integration Server does not augment the instance document.
Is Nillable	Whether the element can have no content in the instance document. The **Is Nillable** value is equal to the value of the `nillable` attribute in the element declaration. The **Is Nillable** field can have the following values: ■ **True** indicates that the `nil` attribute for this element in the instance document can be set to true. (When `nil="true"`, the element has a null value.) When the value of **Is Nillable** is True, the element declaration in the XML Schema contains `nillable="true"`. ■ **False** indicates that the `nil` attribute for the element in an instance document cannot be set to true; that is, the element must contain content. When the value of **Is Nillable** is False, the element declaration contains `nillable="false"` or the nillable attribute is absent. This is the default value.
Is Abstract	Whether the element is abstract. The **Is Abstract** value is equal to the value of the `abstract` attribute in the element declaration. The **Is Abstract** field can have the following values:

This field...	Specifies...
	■ **True** indicates the element is abstract. Abstract element declarations cannot appear in instance documents. Instead, an element in the abstract element's substitution group must appear in the instance document. When **Is Abstract** is True, the element declaration contains `abstract="true"`. ■ **False** indicates the element is not abstract. When the value of **Is Abstract** is False, the element declaration contains `abstract="false"` or the `abstract` attribute is absent. This is the default value.
Complex Type	The name and namespace of the complex type assigned to the element. This field appears only if the element is defined to be of complex type. If the element is defined to be of anonymous complex type, this field displays 'Anonymous' as the name of the complex type. In the Schema Browser, the complex type definition assigned to an element appears as an immediate child of the element.
Simple Type	The name and namespace of the simple type assigned to the element. This field appears only if the element is defined to be of simple type. If the element is defined to be of anonymous simple type, this field displays 'Anonymous' as the name of the simple type. In the Schema Browser, the simple type definition assigned to an element appears as an immediate child of the element.
Min Occurs	The minimum number of times this element must appear. The **Min Occurs** value is equal to the value of the `minOccurs` attribute in the local element declaration. If the local element declaration does not specify `minOccurs`, Designer uses a default value of 1. This field appears only when you select a local element declaration; that is, an element declaration in a complex type definition.
Max Occurs	The maximum number of times this element may appear. The **Max Occurs** value is equal to the value of the `maxOccurs` attribute in the local element declaration. If the local element declaration does not specify `maxOccurs`, Designer uses a default value of 1. This field appears only when you select a local element declaration; that is, an element declaration in a complex type definition.

Element Reference

An element reference is a reference from an element declaration in a content specification to a globally declared element. In a schema generated from an XML Schema, this symbol corresponds to the `ref="`*`globalElementName`*`"` attribute in an `<element>` declaration. In a schema generated from

a DTD, this symbol appears next to an element that is a child of another element; that is, the parent element has only element content

This field...	Specifies...
Name	The local name of the referenced element followed by the namespace to which the referenced element belongs. The **Name** value is equal to the expanded value (prefix plus local name) of the name attribute in the element declaration.
Default	The default value for the referenced element. The **Default** value is equal to the value of the default attribute in the referenced element declaration. During data validation, if the element appears in an instance document but contains no content, Integration Server supplies the element with the **Default** value. If the element does not appear in the instance document, Integration Server does not augment the instance document.
Fixed Value	The fixed value for the referenced element. The **Fixed Value** is equal to the value of the `fixed` attribute in the element declaration. When Integration Server validates an instance document against the schema, if the element appears, its value must be equal to the **Fixed Value**. During data validation, if the element appears in an instance document but contains no content, Integration Server supplies the element with the **Fixed Value**. If the element does not appear in the instance document, Integration Server does not augment the instance document.
Is Nillable	Whether the referenced element can have no content in the instance document. The **Is Nillable** value is equal to the value of the `nillable` attribute in the element declaration. The **Is Nillable** field can have the following values: ■ **True** indicates that the `nil` attribute for this element in the instance document can be set to true. (When `nil="true"`, the element has a null value.) When the value of **Is Nillable** is True, the element declaration in the XML Schema contains `nillable="true"`. ■ **False** indicates that the `nil` attribute for the element in an instance document cannot be set to true; that is, the element must contain content. When the value of **Is Nillable** is False, the element declaration contains `nillable="false"` or the nillable attribute is absent. This is the default value.
Is Abstract	Whether the referenced element is abstract. The **Is Abstract** value is equal to the value of the `abstract` attribute in the element declaration. The **Is Abstract** field can have the following values: ■ **True** indicates the referenced element is abstract. Abstract element declarations cannot appear in instance documents. Instead, an

This field...	Specifies...
	element in the abstract element's substitution group must appear in the instance document. When **Is Abstract** is True, the element declaration contains `abstract="true"`. ■ **False** indicates the referenced element is not abstract. When the value of **Is Abstract** is False, the element declaration contains `abstract="false"` or the `abstract` attribute is absent. This is the default value.
Complex Type	The name and namespace of the complex type assigned to the referenced element. This field appears only if the element is defined to be of complex type. If the element is defined to be of anonymous complex type, this field displays 'Anonymous' as the name of the complex type. In the Schema Browser, the complex type definition assigned to an element appears as an immediate child of the element.
Simple Type	The name and namespace of the simple type assigned to the referenced element. This field appears only if the element is defined to be of simple type. If the element is defined to be of anonymous simple type, this field displays 'Anonymous' as the name of the simple type. In the Schema Browser, the simple type definition assigned to an element appears as an immediate child of the element.
Min Occurs	The minimum number of times this element must appear. The **Min Occurs** value is equal to the value of the `minOccurs` attribute in the local element declaration. If the local element declaration does not specify `minOccurs`, Designer uses a default value of 1. This field appears only when you select a local element declaration; that is, an element declaration in a complex type definition.
Max Occurs	The maximum number of times this element may appear. The **Max Occurs** value is equal to the value of the `maxOccurs` attribute in the local element declaration. If the local element declaration does not specify `maxOccurs`, Designer uses a default value of 1. This field appears only when you select a local element declaration; that is, an element declaration in a complex type definition.

Empty Content

Empty content occurs in XML Schema definition when an element's associated complex type definition does not contain any element declarations. An element with empty content may still carry attributes. In a DTD, an element has empty content when it is declared to be of type EMPTY.

Mixed Content Model

A mixed content model allows character data to be interspersed with child elements. This symbol corresponds to the `mixed="true"` attribute in a complex type definition in an XML Schema definition or a DTD element list in which the first item is #PCDATA.

Sequence Content Model

A sequence content model specifies that the child elements in the instance document must appear in the same order in which they are declared in the content model. This symbol corresponds to the `<sequence>` compositor in an XML Schema definition or a sequence list in an element type declaration in a DTD.

This field...	Specifies...
Min Occurs	The minimum number of occurrences of the content model for an element in the instance document. The value of **Min Occurs** is equal to the value of the `minOccurs` attribute in the `<sequence>` content model
Max Occurs	The maximum number of occurrences of the content model for an element in the instance document. The value of **Max Occurs** is equal to the value of the `maxOccurs` attribute in the `<sequence>` content model.
Summary of Children	The name and occurrence constraints for the child elements in the content model. ■ **Name**. The name of the child element. ■ **Min, Max**. The minimum and maximum occurrence constraints for the child element. The **Min** and **Max** values correspond to the `minOccurs` and `maxOccurs` attributes (respectively) in the local element declaration.

Simple Type Definition

A simple type definition specifies the data type for an element that contains only character data or for an attribute. Unlike complex type definitions, simple type definitions cannot carry attributes. This symbol corresponds to the `<simpleType>` element in an XML Schema definition.

If the simple type definition is unnamed (an anonymous type), the Schema Browser displays 'Anonymous' as the name of the complex type definition.

This field...	Specifies...
Base Constraints	The constraining facet values set in the type definitions from which a simple type was derived. Base constraints are the constraining facet values from the primitive type to the immediate parent type. These constraint values represent the cumulative facet values for the simple type.
Simple Type: *Name*	The local name and target namespace of the simple type. The **Name** value is equal to the expanded value (prefix plus local name) of the name attribute in the type definition. If the Schema Browser displays 'Anonymous' as the name of the simple type, the simple type is an anonymous (unnamed) type defined in an element or attribute declaration.
Primitive Type	The primitive datatype from which the simple type is derived.
constraining facet fields	The constraining facets applied to a simple type definition. This includes fields for values such as enumeration, length, minlength, maxlength, whitespace, and pattern. Which constraining facets are displayed depends on the primitive type of the simple type definition. A constraining facet field contains a value only if the simple type definition specified a value for the facet. The constraining facets fields do not display facet values set in the type definition from which the simple type was derived. For more information about constraining facets, see the W3C specification *XML Schema Part 2: Datatypes*.

Service Properties

To view properties for a service, double-click the service in the Package Navigator of Designer. In the Properties view, you can configure the **Runtime**, **Transient Error Handling**, **Universal Name**, **Audit**, and **Output Template** properties for the service.

Note:
A web service connector also uses the **Universal Name**, **Audit**, and **Output Template** categories of the Properties view, but does not use the **Retry on ISRuntimeException** properties. A web service connector uses all the properties in the **Run time** category with the exception of the **Default xmlFormat** property.

To edit the properties for a service, you must have Write access to it and own the lock.

General Properties for Services

In the Properties view, under **General**, you can assign an ACL to a service.

Note:

General properties for services do not apply to OData services. For more information about the general properties for OData services, see "General Properties for OData Services" on page 998.

Property	Description
Permissions	Click … to assign or view ACL permissions to a service.
Reuse	Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.
Source URI	Displays the location or URI of the source used to create this flow service. A flow service can be created from sources such as XML documents, XML Schema definitions, and WSDL documents. If this flow service was created as an empty flow service and was not based on a source, the **Source URI** property is empty.

Run Time Properties for Services

In the Properties view, under **Run time**, you can specify the following service properties:

- **State of a service.** You can indicate whether or not you want the server to treat it as a "stateless" service at run time
- **Caching of service results.** You can cache elements to reduce memory usage in Integration Server.
- **Execution locale of a service.** You can set the type of locale in which the Integration Server executes at run time.
- **Creating a URL alias for a service.** You can create an alias for the path portion of the URL used to invoke a service.
- **Saving and restoring of the pipeline.** You can save the pipeline or restore a previously saved pipeline at run time.

- **XML format for the service input.** If the service receives an XML document, you can specify the format that Integration Server uses for the document when it passes the document to the service.
- **HTTP methods for a service.** You can select the HTTP methods that can be configured for a service. This selection overrides the HTTP methods configured for a resource corresponding to the service.

Important:
The **Run time** properties in the Properties view should only be set by someone who is thoroughly familiar with the structure and operation of the selected service. Improper use of these options can lead to a service failure at run time and/or the return of invalid data to the client program.

Property	Description
Stateless	Specifies whether or not Integration Server is required to maintain state information for this service for the duration of the client session. Select **True** if the service is a self-contained, atomic unit of work and does not need access to state information. This reduces the number of server resources it consumes at run time. Select **False** if the service is part of a multi-service transaction or if you are unsure of its state requirements. The default is **True**.
Cache results	Indicates whether Integration Server stores the service results in a local cache for the time period specified in the **Cache expire** property. After the service executes, the server places the entire pipeline contents into a local cache. When subsequent requests for the service provide the same set of input values, the server returns the cached results instead of invoking the service again. Select **True** to cache the service results. Select **False** if you do not want to cache service results. Cache results for stateless services only. The default is **False**. **Note:** Caching is only available for data that can be written to the repository. Because XML nodes cannot be written to the repository, they cannot be cached. **Note:** This property relates specifically to the caching of service results described in "About Service Caching" on page 166. This property does not affect caching that is performed by the services in the pub.cache folder.
Cache expire	Specifies the amount of time that the pipeline contents stay in memory after they are cached. If you enable the **Cache results** property, type an integer in this field representing the number of minutes you want a result to remain cached. The expiration timer begins when the server initially

Property	Description
	caches the results. The server does not restart the expiration timer each time it retrieves the results from cache. The minimum cache expiration time is one minute. **Note:** This property relates specifically to the caching of service results described in "About Service Caching" on page 166. This property does not affect caching that is performed by the services in the pub.cache folder.
Reset cache	Click **Reset** to clear the cached results for this service. **Note:** This property relates specifically to the caching of service results described in "About Service Caching" on page 166. This property does not affect caching that is performed by the services in the pub.cache folder.
Prefetch	Determines whether Integration Server automatically refreshes a cached result when it expires by re-executing the service using the same inputs. To automatically refresh this service's cached results, set **Prefetch** to **True**. (**Prefetch** can consume a substantial amount of server memory; consult your Server Administrator before using this option.) The cache may not be refreshed at the exact time specified in **Cache expire**. It may vary from 0 to 15 seconds, according to the cache sweeper thread. For details, see the watt.server.cache.flushMins setting in Integration Server. **Note:** This property relates specifically to the caching of service results described in "About Service Caching" on page 166. This property does not affect caching that is performed by the services in the pub.cache folder.
Prefetch activation	Specifies the minimum number of times that a cached result must be accessed (hit) with the same inputs in order for the server to prefetch results when it expires. If you enable **Prefetch**, you must specify an integer representing the minimum number of hits a cached result must receive to be eligible for prefetch. (Entries that do not receive the minimum number hits are released from memory.) **Note:** The cache may not be refreshed at the exact time the last hit fulfills the **Prefetch Activation** requirement. It may vary from 0 to 15 seconds, according to the cache sweeper thread. For details, see the watt.server.cache.flushMins setting in Integration Server.

Property	Description
	Note: This property relates specifically to the caching of service results described in "About Service Caching" on page 166. This property does not affect caching that is performed by the services in the pub.cache folder.
Execution locale	Specifies the locale in which this service will be executed.
HTTP URL Alias	Specifies an alias for the path portion of the URL used to invoke a service. For a flow service, the path portion of the URL consists of the invoke directive and the fully qualified service name. For a REST service, the path portion of the URL consists of the rest directive and the location of the REST resource folder in which the service resides.
Pipeline debug	Determines whether Integration Server automatically saves or restores the pipeline after the service executes. This option is useful for testing and debugging the service. **Select... / To...** **None** — Run the service without saving or restoring the pipeline. This is the default. **Save** — Save the entire pipeline contents to a file when the service executes. **Restore (Override)** — To restore the pipeline from a file when the service executes. **Restore (Merge)** — To merge the pipeline with one from a file when the service executes. When this option is selected and the input parameters in the file match the input parameters in the pipeline, the values defined in the file are used in the pipeline. If there are input parameters in the pipeline that are not matched in the file, the input parameters in the pipeline remain in the pipeline. **Note:** The options you select can be overwritten at run time by the value of the watt.server.pipeline.processor property, set in the server configuration file. This property specifies whether to globally enable or disable the Pipeline debug feature. The default enables the Pipeline debug feature on a service-by-service basis. For more information on setting properties in the server configuration file, see *webMethods Integration Server Administrator's Guide*.

Property	Description
	Note: When using MTOM streaming for SOAP attachments, *messageContext* variables and/or XOPObject fields will not be available in the saved pipeline. A *messageContext* variable is used by many pub.soap services to hold the SOAP message on which the service acts. XOPObject fields are Objects that use the com.wm.util.XOPObject Java wrapper type. For more information about MTOM Streaming, see the *Web Services Developer's Guide*.
Default xmlFormat	The default XML format for XML documents received by the service. **Note:** You can specify the default XML format for flow services and Java services only. The **Default xmlFormat** property is not available for C/C++ services, .NET services, or web service connectors.

Select...	To...
<blank>	Specify that Integration Server obtains the default XML format from the watt.server.http.xmlFormat server configuration parameter. This is the default. For more information about the watt.server.http.xmlFormat server configuration parameter, see *webMethods Integration Server Administrator's Guide*.
bytes	Specify that Integration Server uses a byte array as the default XML format. Integration Server passes the XML document directly to the service as a byte array without parsing the XML. Integration Server places the byte array in the input pipeline of the target service in a variable named *xmlBytes*.
enhanced	Specify that Integration Server uses a node parsed by the enhanced XML parser as the default XML format. Integration Server parses the XML automatically using the enhanced XML parser. Integration Server uses the default options specified for enhanced XML parsing on the **Settings > Enhanced XML Parsing** page in Integration Server Administrator. Integration Server passes the XML document to the target service as a node that implements the w3c.com.Node interface. Integration Server places the node in the input pipeline of the target service in a variable named *node*. **Note:**Integration Server writes the XML document to a cache during parsing only if caching is configured for enhanced XML parsing. Integration Server uses BigMemory while parsing only if caching is configured and BigMemory is enabled for use with enhanced XML parsing. For more

Property	Description
	information about configuring a cache andBigMemory for XML parsing, see *webMethods Integration Server Administrator's Guide*.
	node Specify that Integration Server uses a node parsed by the legacy XML parser as the default XML format. Integration Server parses the XML automatically using the legacy parser and passes it to the target service as a node. Integration Server places the node in the input pipeline of the target service in a variable named *node*
	stream Specify that Integration Server uses an InputStream as the default XML format. Integration Server passes the XML document directly to the service as an XML stream without parsing the XML. Integration Server places the XML stream in the input pipeline of the target service as an InputStream named *xmlStream*.
Allowed HTTP methods	Click … to select the HTTP methods that you can configure for a service. The supported methods are GET, HEAD, PUT, POST, PATCH, DELETE, and OPTIONS. **Important:** If the service already has REST resources configured, Designer displays a warning message if you change the selection of the allowed HTTP methods to exclude any method used in the configuration of the resources. In such a situation, any client request invoking the excluded method will fail. Therefore, you must ensure that the set of HTTP methods configured for a REST resource is always a subset of the methods allowed for the underlying service.

Transient Error Handling Properties

When building a service, you can specify what action Integration Server takes when the service fails because of a transient error caused by a run-time exception. That is, you can specify whether or not Integration Server should retry the service.

In the Properties view, under **Transient error handling**, you specify whether or not Integration Server should retry the service.

Property	Description
Max retry attempts	Specifies the number of times Integration Server should attempt to re-execute the service when the service fails because of an ISRuntimeException. An

Property	Description
	ISRuntimeException occurs when the service catches a transient error, wraps the error, and re-throws it as an exception. (A *transient error* is an error that arises from a condition that might be resolved quickly, such as the unavailability of a resource due to network issues or failure to connect to a database.) The default is 0, which indicates that Integration Server does not attempt to re-execute the service.
Retry interval	Specifies the number of milliseconds that Integration Server should wait between retry attempts. The default is 0 milliseconds, which indicates that Integration Server re-executes the service immediately **Note:**Integration Server enforces a *maximum retry period* when you configure service retry properties. The maximum retry period indicates the total amount of time that can elapse if the Integration Server makes the maximum retry attempts. By default, the maximum retry period is 15,000 milliseconds (15 seconds). When you configure service retry, Integration Server verifies that the retry period for that service will not exceed the maximum retry period. Integration Server determines the retry period for the service by multiplying the maximum retry attempts by the retry interval. If this value exceeds the maximum retry period, Designer displays an error indicating that either the maximum attempts or the retry interval needs to be modified.

Audit Properties

In the Properties view, under **Audit**, you enable auditing and specify when a service should generate audit data.

Property	Description	
Enable auditing	Specifies when the service generates audit data	
	Select...	**To...**
	Never	Indicate that the service should never generate audit data. Select this option if you do not want to be able to audit this service.
	When top-level service only	Indicate that the service should generate audit data only when it is invoked by a client request or a trigger. The service does not generate audit data when it is invoked by another service (that is, when it is a nested service).
	Always	Indicate that the service should generate audit data every time it executes. Select this option if the service is a critical

Property	Description
	service that you want to be able to audit every time it executes.
Log on	Specifies the execution points at which the service generates audit data.

Select...	To...
Error only	Indicate that the service should generate audit data only when the service fails. If the service executes successfully, it will not generate audit data. **Performance Impact:** This option impacts performance only when the service fails. When a service executes successfully, this option does not impact performance. This option offers the smallest performance impact of all the options under **Log on**.
Error and success	Indicate that the service should generate audit data when the service finishes executing. The service will generate audit data regardless of whether it ends because of success or failure. The service log will contain an entry for every time the service finishes processing. **Performance Impact:** This option impacts performance every time the service executes, whether it ends because of error or success. If you are concerned only with services that fail, consider using the **Error only** option instead.
Error, success, and start	Indicate that the service should generate audit data when it begins executing and when it finishes executing. The service will generate audit data twice every time it executes (once when it begins processing and once when it finishes processing). Generally, most services execute fairly quickly. By the time an administrator views the service log using webMethods Monitor, the service log would probably contain entries for the start and end of service execution. Situations where you might want the service to generate audit data at the start and end of service execution include: ■ To check for the start of long-running services ■ To detect service hangs. In both situations, if service execution began but did not complete, the service log contains an entry for the start of the service, but no entry for the end of the service.

Property	Description
	Performance Impact: Of all the options under **Log on**, this option provides the most verbose and expensive type of audit logging. Every time it executes, the service generates audit data at two points: the beginning and the end. Integration Server must write the audit data to the service log twice per service execution. This requires significantly more disk utilization than the **Error only** and **Error and success** options. At most, the **Error only** and **Error and success** options require Integration Server to write audit data once per service execution.
Include pipeline	Specifies when Integration Server should include a copy of the input pipeline in the service log.

Select...	To...
Never	Indicate that the input pipeline should never be included in the service log. Select this option if you are using a flat file for the service log or if you do not want to be able to resubmit the service to Integration Server. **Performance Impact:** This option requires minimal network bandwidth because Integration Server needs to send only the audit data generated by the service to the service log.
On errors only	Indicate that the input pipeline should be included in the service log only when the service ends because of errors. Select this option if you want to use the resubmission capabilities of webMethods Monitor to reinvoke a failed service. For more information about webMethods Monitor, see the webMethods Monitor documentation. **Performance Impact:** For successful service invocations, the **On errors only** option requires minimal network bandwidth. Service invocations that end in failure require more network bandwidth because the Integration Server must save the audit data and the input pipeline. The actual network bandwidth needed depends on the size of the initial input pipeline. A large pipeline can degrade performance because it may negatively impact the rate at which the data is saved to the service log.
Always	Indicate that Integration Server saves a copy of the input pipeline to the service log every time the service generates audit data. If the service generates data at the start and end of execution (**Log on** is set to **Error, success, and start**), the input pipeline is saved with the service log entry for the start of service execution. If a service does not generate

Property	Description
	audit data, Integration Server does not include a copy of the input pipeline. Select the **Always** option if you want to be able to use the resubmission capabilities of the webMethods Monitor to reinvoke the service, regardless of whether the original service invocation succeeded or failed. Including the pipeline can be useful if a resource experiences a fatal failure (such as hard disk failure). To restore the resource to its pre-failure state, you could resubmit all the service invocations that occurred since the last time the resource was backed up. This is sometimes called a full audit for recovery. **Performance Impact:** The **Always** option is the most expensive option under **Include pipeline**. This option places the greatest demand on network bandwidth because Integration Server must write a copy of the input pipeline to the service log every time a service executes. The actual network bandwidth needed depends on the size of the initial input pipeline. A large input pipeline can negatively impact the rate at which the data is saved to the service log. **Note:** If you want audit events generated by this service to pass a copy of the input pipeline to any subscribed event handlers, select **On errors only** or **Always**.

Important:
The options you select can be overwritten at run time by the value of the watt.server.auditLog server property, set in the server configuration file. This property specifies whether to globally enable or disable service logging. The default enables customized logging on a service-by-service basis.

Circuit Breaker Properties

You can configure a circuit breaker to act as a proxy for executing the service. A circuit breaker can detect errors or inordinately long execution times for a service and prevent execution of the failing or slow service by subsequent service calls.

In the Properties view, under **Circuit Breaker**, you enable the circuit breaker and configure its behavior.

Note:

To use the circuit breaker feature with Integration Server, your Integration Server must have additional licensing. In addition to the licensing requirement, to use the circuit breaker functionality in version 10.1, you must install the following fixes: ESB_10.1_Fix2 and IS_10.1_Core_Fix2.

Property	Description
Enabled	Specifies whether a circuit breaker is used with the service. **Specify...** — **To...** **True** — Enable a circuit breaker for the service. **False** — Disable a circuit breaker for the service. This is the default.
Failure event	Specifies what constitutes a failure event. Specify one of the following **Specify...** — **To...** **Exception only** — Indicate that a failure event occurs only when the service ends with an exception. This is the default. **Timeout only** — Indicate that a failure event occurs only when the service execution time exceeds the **Timeout period** property value. **Exception or Timeout** — Indicate that a failure event occurs when the service ends with an exception or the service execution time exceeds the **Timeout period** property.
Timeout period	Specifies the number of seconds that service execution can take before being considered a timeout failure event. If the timeout period elapses before service execution completes, the circuit breaker considers a timeout failure event to have occurred. The default is 60 seconds. You must specify a **Timeout period** greater than 0 if you set the **Failure Event** property to **Timeout** or **Exception or timeout**. If you set **Failure event** to **Exception**, the **Timeout period** property is display only. Whether or not the circuit breaker attempts to end service execution when the execution time meets or exceeds the

Property	Description
	timeout period depends on the value of the **Cancel thread on timeout period** property.
Cancel thread on Timeout	Specifies whether the circuit breaker gracefully attempts to cancel the thread executing the service when the timeout period elapses causing the timeout failure event. Canceling a thread can free up resources held by the thread. For circuit breaker to cancel a thread, the watt.server.threadKill.enabled property must be set to true. If you want circuit breaker to attempt to interrupt a service thread in addition to attempting to cancel it, the watt.server.threadKill.interruptThread.enabled property must be set to true. Use care when configuring a circuit breaker to cancel threads. Canceling a thread might not free up resources being held by the service. For more information about canceling threads, see the *webMethods Integration Server Administrator's Guide*.
	Specify... / **To...** **True** — Indicate that circuit breaker attempts to cancel the thread executing the service if the timeout period elapses before the service execution completes. **False** — Indicate that circuit breaker does not attempt to cancel the service execution thread if the timeout period elapses before service execution completes. This is the default.
Failure threshold	The number of failure events occurring within the failure period that cause the circuit to open. The default is 5. If circuit breaker is enabled for the service, you must specify a value greater than 0.
Failure period	Time period, measured in seconds, during which the number of failure events meeting or exceeding the failure threshold causes the circuit breaker to open the circuit. The default is 60 seconds.
Circuit open action	Action the circuit breaker takes when receiving requests to invoke the service when the circuit is open.

Property	Description	
	Specify...	**To...**
	Throw exception	Indicate that circuit breaker throws the exception that caused the circuit to open This is the default.
	Invoke service	Indicate that circuit breaker invokes the service specified in the **Circuit open service** property.
Circuit open service	Fully qualified name of the service that circuit breaker invokes when receiving requests for this service when the circuit is open. This property applies only if the **Circuit open action** property is set to **Invoke service**. The **Circuit open service** property is display-only if the **Circuit open action** property is set to **Throw exception**.	
Circuit reset period	Length of time, measured in seconds, for which the circuit remains in an open state once it is opened. During the reset period, the circuit breaker responds to requests to invoke the service as specified by the **Circuit open action** property. When the reset period elapses, the circuit breaker places the circuit in a half-open state. The next request for the service results in service execution, after which the circuit breaker either closes or re-opens the circuit. The default is 300 seconds.	

Universal Name Properties for Services

You can specify a unique public identifier that external protocols (such as SOAP) use to reference a service on an Integration Server. Every service on an Integration Server has an explicit universal name in addition to its regular implicit webMethods name. If you omit or delete a service's explicit universal name, it still retains its implicit universal name.

In the Properties view, under **Universal Name**, you assign a universal name to a service.

A universal name has two parts: a *namespace name* and a *local name*.

Property	Description
Namespace name	Specifies the name used to qualify the local name of this service. The namespace name you specify must be a valid absolute URI (relative URIs are not supported).

Property	Description
Local name	Specifies a name that uniquely identifies this service within the collection encompassed by **Namespace name**. The name can be composed of any combination of letters, digits, or the period (.), dash (-), or underscore (_) characters. The name must begin with a letter or the underscore character. **Note:** Most webMethods users use the unqualified portion of the service name as the local name.

Output Template Properties for Services

In the Properties view, under **Output template**, you can assign an output template to a service.

Property	Description
Name	Specifies the name of the file that contains the output template for the selected service. To assign an *existing* template to this service, type the name of the template file in this field. To create a *new* template file for this service, type a name for the template in this field.
Template source	Opens the Template source page so that you can edit the existing output template. **Note:** Changes you make to an output template are written to the template file when you click **Save** in the Template source page. Changes that you make to a template file affect *all* the services in the package that use the template, not just the service that is currently open in the editor.

Specification Properties

In the Properties view, you can set the properties for a specification. To view the properties for a specification, double-click the specification in Package Navigator view.

To edit the properties for a specification, you must have Write access to it and own the lock.

General Properties for Specifications

In the Properties view, under **General**, you can assign list or write ACL privileges to a specification.

Property	Description
Permissions	Click … to assign or view ACL permissions to a specification.

Property	Description
Reuse	Specifies whether this specification can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.

Transformer Properties

In the Properties view, you can set the properties for a transformer inserted into a MAP step.

To view properties for a transformer, double-click the transformer in the Pipeline view of Designer.

General Properties for Transformers

In the Properties view, under **General**, you can view and configure the service and validations properties for a transformer.

Property	Description
Service	Specifies the fully qualified name of the service that is invoked at run time. When you insert a transformer, Designer automatically assigns the name of that service to the **Service** property. If the service that a transformer invokes is moved, renamed, or deleted, you must change the **Service** property. Specify the service's fully qualified name in the *folderName:serviceName* format or click … to select a service from a list.
Validate input	Specifies whether or not Integration Server validates the input to the transformer against the input signature of the service. Select **True** if you want to validate the input of the service. Select **False** if you do not want to validate the input of the service.
Validate output	Specifies whether or not Integration Server validates the output of the transformer against the output signature of the service. Select **True** if you want to validate the output of the service. Select **False** if you do not want to validate the output of the service.

Variable Properties

You can specify the data type and input values for a variable. You can also apply content constraints and structural constraints to a variable for validation purposes. A variable can be a String, String list, String table, document, document list, document reference, document reference list, Object, or Object list.

In the Properties view, select a variable in the editor to set general properties and constraints for the variable.

Note:
Specific properties in the Properties view are enabled or disabled, depending on the type of variable you have selected.

General Properties for Variables

In the Properties view, under **General**, you can change the data type for a variable. You can also associate the variable with an XML namespace and specify input values and an input method.

Property	Description
Name	Specifies the name of the variable. To change the name of a variable, rename it in the editor.
Type	Specifies the data type of the variable.
XML namespace	Specifies the XML namespace to which the variable belongs.
Comments	Specifies a comment about the variable.
Model type	Specifies the content model for a document or document list variable. The content model provides a formal description of the structure and allowed content for a document. The **Model type** property is display-only. To change the model type for a document or document list, modify the corresponding complex type definition in the XML schema definition, and recreate the document type that contains this document or document list. The contents of a document or document list variable with a **Model type** property value other than “Unordered” cannot be modified. The **Model type** property can have one of the following values:

Value...	Description...
All	At run time, all of the fields in the document must appear once or not at all, and in any order. The all model type corresponds to an complex type definition that contains an all compositor in the model group.

Property	Description	
	Choice	One and only one of the fields in the document can appear. The choice model type corresponds to an complex type definition that contains a choice compositor in the model group.
	Sequence	Fields in the instance document must appear in the same order in which they are declared in the document type. The sequence model type corresponds to an complex type definition that contains a sequence compositor in the model group.
	Simple	At run time, the document contains a single field that contains simple content and carries an attribute. The document field contains a *@attributeName* field for the attribute value and a **body* field for the simple content.
	Unordered	At run time, fields in the document can appear in any order and any number of times. **Note:** An unordered data structure is sometimes referred to as a "bag" data structure.
String display type	Specifies how you want to enter input data for this variable. You can only select a display type if the variable is a String. Select one of the following:	
	Select...	**To...**
	Text Field	Enter the input in a text field.
	Password	Enter the input as a password, where asterisks indicate the input instead of characters.
	Large Editor	Enter the input in a large text area instead of a text field. This is useful if you expect a large amount of text as input for the variable, or if you need to have TAB or new line characters in the input.
	Pick List	Limit the input to a predefined list of values. These values appear as choices when Designer prompts for input at run time. Click the **Pick list choices** property's **Edit** button to specify the list of values you want users to select from.
Pick list choices	Allows you to enter the list of values that users can select for this variable.	
Document reference	For a document reference or document reference list, this property specifies the fully qualified name of the document type in the Package Navigator view that the variable references.	

Property	Description
Substitution group	If the variable represents an element that is a member of a substitution group, identifies the head element for which this element can be substituted.

Constraints Properties for a Variable

Use these properties to apply structural and content constraints to the variable. You only need to specify constraints if you plan to use this variable with validation.

Property	Description
Min occurs	Specifies the minimum number of occurrences of the variable at run time. This property is display-only. **Note:** The **Min occurs** property appears only for a variable in a document type with a **Model type** property value of Sequence, Choice, or All.
Max occurs	Specifies the maximum number of occurrences of the variable at run time. A value of "unbounded" indicates that the variable can appear an unlimited number of times. This property is display-only. **Note:** The **Max occurs** property appears only for a variable in a document type with a **Model type** property value of Sequence, Choice, or All.
Required	Specifies whether or not the variable needs to exist at run time. The **Required** property appears for variables in document types if one or more of the following are true: ■ The document type was created using a version of Integration Server prior to version 8.2. ■ The document type was created using Developer. ■ The **Model type** property of the document type is Unordered.

Select...	To...
True	The default value. Specifies that the variable must exist in the pipeline at run time.
False	Specifies that the existence of the variable is optional at run time.

Allow null	Specifies whether null is a valid value for this variable.

Property	Description	
	Select...	**To...**
	True	The default value. Specifies that a null value will not result in a validation error at run time.
	False	Specifies that a null value will cause a validation error at run time.
Allow unspecified fields	Specifies whether the document is open or closed. This property is enabled only if the variable is a document or document list.	
	Select...	**To...**
	True	The default value. The document is considered open. At run time, fields that exist in the document but are not declared in the document field will not cause errors.
	False	The document or document list is considered closed. At run time, fields that exist in the document but are not declared in the document field will be treated as errors.
Content type	Specifies the XML schema simple type that constrains the value of the **String** field. This property is enabled if the variable is a String, String list, or String table. To view and edit the content constraint for a variable, click … and select one of the following:	
	Select...	**To...**
	Content type	Select a content constraint from the drop-down menu that corresponds to a simple type built-in to XML Schema.
	Browse	Use a simple type from an IS schema as the content constraint.
	Customize	Customize a simple type by modifying the constraining facets.
	Base constraints	View the constraining facet values set in the type definitions from which a simple type was derived.
Java wrapper type	Specifies the Java class of an **Object** field. This property is enabled if the variable is an Object or Object list.	

Constraints Applied to Variables

Designer displays small symbols next to a variable icon to indicate the constraints applied to the variable. Designer displays variables in the following ways:

Variable	Constraint status	Variable Properties
String	Required field.	The Required property is set to True.
String	Optional field.	The Required property is set to False.
String	Required field with content type constraint	The Content type property specifies an IS schema or XML schema.
String	Optional field with content type constraint	The Required property is set to False and the Content type property specifies an IS schema or XML schema.

Note: Designer displays the ‡ symbol next to String, String list, and String table variables with a content type constraint only. Designer does not display the ‡ symbol next to Object and Object list variables with a specified Java class constraint. Object and Object lists with an applied Java class constraint have a unique icon. For more information about icons for constrained Objects, see "Java Classes for Objects" on page 1085.

Web Service Connector Properties

To view properties for a web service connector, double-click the web service connector in the Package Navigator of Designer. In the Properties view, under **Web Service Properties**, you can configure the **Runtime**, **Universal Name**, **Audit**, and **Output Template** properties for the service.

To edit the properties for a web service connector, you must have Write access to it and own the lock.

General Properties for Web Service Connectors

In the Properties view, under **General**, you can assign an ACL to a web service connector.

Property	Description
Permissions	Click … to assign ACL permissions to a service.
Reuse	Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's

Property	Description
	status in CentraSite will not change until the next publication of assets to CentraSite.
Source URI	Displays the location of the source WSDL used to create the web service connector.

Run Time Properties

In the Properties view, under **Run time**, you can specify the following web service connector parameters:

- **State of a service.** You can maintain whether or not you want the server to treat it as a "stateless" service at run time.
- **Caching of service results.** You can cache elements to reduce memory usage in Integration Server.
- **Execution locale of a service.** You can set the type of locale in which the Integration Server executes at run time
- **Saving and restoring of the pipeline.** You can save the pipeline or restore a previously saved pipeline at run time.

Important:
The Run time properties in the Properties view should only be set by someone who is thoroughly familiar with the structure and operation of the selected service. Improper use of these options can lead to a service failure at run time and/or the return of invalid data to the client program.

Property	Description
Stateless	Specifies whether or not Integration Server is required to maintain state information for this service for the duration of the client session. Select **True** if the service is a self-contained, atomic unit of work and does not need access to state information. This reduces the number of server resources it consumes at run time. Select **False** if the service is part of a multi-service transaction or if you are unsure of its state requirements. The default is **False**.
Cache results	Indicates whether Integration Server stores the service results in a local cache for the time period specified in the **Cache expire** property. After the service executes, the server places the entire pipeline contents into a local cache. When subsequent requests for the service provide the same set of input values, the server returns the cached results instead of invoking the service again. Select **True** to cache the service results. Select **False** if you do not want to cache service results. Cache results for stateless services only. The default is **False**.

Property	Description
	Note: Caching is only available for data that can be written to the repository server. Because XML nodes cannot be written to the repository, they cannot be cached.
Cache expire	Specifies the amount of time that the pipeline contents stay in memory after they are cached. If you enable the **Cache results** property, type an integer in this field representing the number of minutes you want a result to remain cached. The expiration timer begins when the server initially caches the results. The server does not restart the expiration timer each time it retrieves the results from cache. The minimum cache expiration time is one minute.
Reset cache	Click **Reset** to clear the cached results for this service
Prefetch	Determines whether Integration Server automatically refreshes a cached result when it expires by re-executing the service using the same inputs. To automatically refresh this service's cached results, set **Prefetch** to **True**. (**Prefetch** can consume a substantial amount of server memory; consult your Server Administrator before using this option.) The cache may not be refreshed at the exact time specified in **Cache expire**. It may vary from 0 to 15 seconds, according to the cache sweeper thread. For details, see the watt.server.cache.flushMins setting in Integration Server.
Prefetch activation	Specifies the minimum number of times that a cached result must be accessed (hit) with the same inputs in order for the server to prefetch results when it expires. If you enable **Prefetch**, you must specify an integer representing the minimum number of hits a cached result must receive to be eligible for prefetch. (Entries that do not receive the minimum number hits are released from memory.) The cache may not be refreshed at the exact time the last hit fulfills the **Prefetch Activation** requirement. It may vary from 0 to 15 seconds, according to the cache sweeper thread. For details, see the watt.server.cache.flushMins setting in Integration Server.
Execution locale	Specifies the locale in which this service will be executed.
Pipeline debug	Determines whether Integration Server automatically saves or restores the pipeline after the service executes. This option is useful for testing and debugging the service. Select **None** to run the service without saving or restoring the pipeline. This is the default. Select **Save** if you want to save the entire pipeline contents to a file when the service executes.

Property	Description
	To restore the pipeline from a file when the service executes, choose **Restore (Override)**. If you want the server to merge the pipeline with one from a file when the service executes, choose **Restore (Merge)**. When this option is selected and the input parameters in the file match the input parameters in the pipeline, the values defined in the file are used in the pipeline. If there are input parameters in the pipeline that are not matched in the file, the input parameters in the pipeline remain in the pipeline. **Note:** The options you select can be overwritten at run time by the value of the watt.server.pipeline.processor property, set in the server configuration file. This property specifies whether to globally enable or disable the Pipeline debug feature. The default enables the Pipeline debug feature on a service-by-service basis. For more information on setting properties in the server configuration file, see *webMethods Integration Server Administrator's Guide*. **Note:** When using MTOM streaming for SOAP attachments, *messageContext* variables and/or XOPObject fields will not be available in the saved pipeline. A *messageContext* variable is used by many `pub.soap` services to hold the SOAP message on which the service acts. XOPObject fields are Objects that use the com.wm.util.XOPObject Java wrapper type. For more information about MTOM Streaming, see the *Web Services Developer's Guide*.

Audit Properties

In the Properties view, under **Audit**, you enable auditing and specify when a service should generate audit data.

Property	Description	
Enable auditing	Specifies when the service generates audit data	
	Select...	**To...**
	Never	Indicate that the service should never generate audit data.
	When top-level service only	Indicate that the service should generate audit data only when it is invoked by a client request or a trigger.
	Always	Indicate that the service should generate audit data every time it executes.
Log on	Specifies the execution points at which the service generates audit data.	

Property	Description	
	Select...	**To...**
	Error only	Indicate that the service should generate audit data only when the service fails.
	Error and success	Indicate that the service should generate audit data when the service finishes executing. The service will generate audit data regardless of whether it ends because of success or failure.
	Error, success, and start	Indicate that the service should generate audit data when it begins executing and when it finishes executing. The service will generate audit data twice every time it executes (once when it begins processing and once when it finishes processing).
Include pipeline	Specifies when Integration Server should include a copy of the input pipeline in the service log.	
	Select...	**To...**
	Never	Indicate that the input pipeline should never be included in the service log.
	On errors only	Indicate that the input pipeline should be included in the service log only when the service ends because of errors.
	Always	Indicate that the input pipeline should always be included in the service log.

Note:
If you want audit events generated by this service to pass a copy of the input pipeline to any subscribed event handlers, select **On errors only** or **Always**.

Important:
The options you select can be overwritten at run time by the value of the watt.server.auditLog server property, set in the server configuration file. This property specifies whether to globally enable or disable service logging. The default enables customized logging on a service-by-service basis.

Universal Name Properties

Specifies a unique public identifier that external protocols (such as SOAP) use to reference a service on an Integration Server. Every service on an Integration Server has an explicit universal name in addition to its regular implicit webMethods name. If you omit or delete a service's explicit universal name, it still retains its implicit universal name.

A universal name has two parts: a *namespace name* and a *local name*.

Property	Description
Namespace name	Specifies the name used to qualify the local name of this service. The namespace name you specify must be a valid absolute URI (relative URIs are not supported).
Local name	Specifies a name that uniquely identifies this service within the collection encompassed by **Namespace name**. The name can be composed of any combination of letters, digits, or the period (.), dash (-), or underscore (_) characters. The name must begin with a letter or the underscore character. **Note:** Most webMethods users use the unqualified portion of the service name as the local name.

Output Template Properties

In the Properties view, under **Output template**, you can assign an output template to a web service connector.

Property	Description
Name	Specifies the name of the file that contains the output template for the selected service. To assign an *existing* template to this service, type the name of the template file in this field. To create a *new* template file for this service, type a name for the template in this field.
Template source	Opens the Template source page so that you can edit the existing output template. **Note:** Changes you make to an output template are written to the template file when you click **Save** in the Template source page. Changes that you make to a template file affect *all* the services in the package that use the template, not just the service that is currently open in the editor.

Web Service Descriptor Properties

To view properties for a provider or consumer web service descriptor, double-click the web service descriptor in Package Navigator view of Designer.

General Properties for Web Service Descriptors

In the Properties view, under **General**, you can view and assign properties to a web service descriptor.

Property	Description
Name	Displays the name of the web service descriptor.
Direction	Displays whether the web service descriptor is for a provider web service (that can be invoked by an external user) or for a consumer web service (that requests the use of a provider entity's web service).
WS-I compliance	Specifies whether the web service descriptor enforces compliance with the WS-I Basic Profile 1.1 standards. **Note:** WS-I Basic Profile 1.0 supports only HTTP or HTTPS bindings. Consequently, WS-I compliance cannot be enforced if the WSDL contains a SOAP over JMS binding. The **WS-I compliance** property cannot be set to true if a web service descriptor has a JMS binder.
Reuse	Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project. When this property is set to public, you can drag the asset to a BPM process or CAF project. When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project. All published assets are available for Impact Analysis, whether they are public or private. Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's status in CentraSite will not change until the next publication of assets to CentraSite.
Source URI	Displays the location of the source used to create the web service descriptor. For a consumer web service descriptor or a WSDL first provider web service descriptor, the **Source URI** is the location of the WSDL document. For a service first web service descriptor, the **Source URI** is empty.
Target namespace	Displays the XML Target Namespace of the web service. By default this is set to the fully qualified URL of the host server.
WSDL URL	URL used to retrieve the WSDL for the web service.
Namespaces	Displays a list of the XML namespaces and the associated namespace prefixes used within the web service descriptor when it was initially created.
Attachment enabled	Specifies whether any attachment should be enabled. The default is **False**.
Pipeline headers enabled	Specifies whether the contents of the SOAP header are placed in the pipeline as a document named *soapHeaders*.

Property	Description
	When this property is set to true for a provider web service descriptor and an IS service that corresponds to an operation in the WSD is invoked, Integration Server places the contents of the SOAP request header in the input pipeline for the IS service. When this property is set to true for a consumer web service descriptor and one of the web service connectors is invoked, Integration Server places the contents of the SOAP response header in the output pipeline for the web service connector. The default is **False**. **Note:** For web service descriptors contained in packages created in earlier versions of Integration Server, the **Pipeline headers enabled** property is set to true.
Validate SOAP response	For a consumer web service descriptor, specifies whether Integration Server validates a SOAP response received by any web service connectors within the consumer WSD. The default is **True**. **Note:** This property applies to consumer web service descriptors only.
Created on version	Identifies the version of Integration Server on which the web service descriptor was created.
Pre-8.2 compatibility mode	Indicates whether or not the web service descriptor runs in pre-8.2 compatibility mode. Web service descriptors that run in pre-8.2 compatibility mode are compatible with versions of Integration Server prior to version 8.2. web service descriptors that do not run in pre-8.2 compatibility mode are compatible with versions of Integration Server 8.2 and later. For web service descriptors created using Designer on Integration Server 8.2 and later, the default is **False**. For web service descriptors created using Developer, the default is **True**.
Validate schemas using Xerces	Indicates whether Integration Server validates the schemas associated with the web service descriptor in the following situations: ■ You change the IS schemas, document types, or signatures of the services associated with a web service descriptor. ■ You refresh the web service connectors for a consumer web service descriptor. The default is **True**. If you migrated a web service descriptor from a previous version of Integration Server, the migration utility set the value based on the version of Integration Server from which the web service descriptor was migrated.

Property	Description
	■ If the web service descriptor was migrated from Integration Server version 7.1.x, the migration utility sets the **Validate Schema using Xerces** property to true. ■ If the web service descriptor was migrated from Integration Server version 8.x, the migration utility used the value of the watt.server.wsdl.validateWSDLSchemaUsingXerces parameter to determine the value of the **Validate Schema using Xerces** property. If the parameter is set to true, the migration utility set the property to true. It the parameter is set to false, the migration utility set the property to false. **Note:** The watt.server.wsdl.validateWSDLSchemaUsingXerces parameter was removed in Integration Server version 9.0.
Outbound callback service	Fully qualified name of the IS service that Integration Server must invoke for an outbound SOAP message if you want to insert custom processing logic into a SOAP request message in case of a consumer web service descriptor and a SOAP response message in case of a provider web service descriptor. For more information about outbound callback services, see *Web Services Developer's Guide*.
Filter login credentials	Indicates whether or not Integration Server filters the login credentials in incoming SOAP requests based on the credentials that are provided in the WS-Security policy attached to the web service descriptor. When this property is set to true, Integration Server filters the login credentials in incoming SOAP requests and processes only those credentials that are provided in the WS-Security policy attached to the web service descriptor. When this property is set to false, Integration Server processes all the credentials that are available in the incoming SOAP request without verifying whether the credentials are also provided in the WS-Security policy attached to the web service descriptor. The default is **True**. Integration Server applies this property to process incoming requests in case of provider web service descriptors and to process asynchronous responses in case of consumer web service descriptors.
Omit xsd:any from WSDL	When generating the schema definition in a WSDL for a provider web service descriptor, specifies whether the xsd:any element is omitted from the complex type definition that corresponds to an open document. Integration Server considers a document to be open if the **Allow unspecified fields** property is set to **True** and considers a document to be closed if the **Allow unspecified fields** property is set to **False**.

Property	Description	
	Select...	**To...**
	True	Omit the xsd:any element from the schema portion of the WSDL document whether or not the document variable has **Allow unspecified fields** set to true.
	False	Retain the xsd:any element in the schema portion of the WSDL document only if the document variable has **Allow unspecified fields** set to true.

Note:
For changes to this property to take effect, save the changes and either refresh the web service descriptor or reload the package that contains the web service descriptor.

Web Service Descriptor Operation Properties

The Properties view displays basic information about the web service operation (or about the header, body, and fault documents) when the operation or one of its documents is selected in the web service descriptor editor.

Note:
Selecting the Response or Request element in an operation simply displays the properties for the operation as a whole. You must select the individual Header, Body, or Fault to display its properties.

Operation Properties

In the Properties view, under **General**, you can view basic information about an operation in the web service descriptor.

Property	Description
Operation Name	Displays the name of the operation. For a provider web service descriptor, this will be the local portion of the Universal Name of the IS service. For a consumer web service descriptor, this will be the operation Name from the WSDL that was used to create the web service descriptor.
IS Service	Displays the fully qualified name of the IS service representing this operation.

Body Element Properties

In the Properties view, under **General**, you can view basic information about the Body element in an operation's request or response. None of the properties of a Body element can be modified.

Property	Description
Name	Name of the Body element.
Document type	Fully qualified name of the IS Document type that defines the body.
Namespace owner	Namespace owner of the Body element.
Signature type	Signature type of the Body element.
Schema URL	URL to the XML schema definition if the signature source is an element declaration from an XML schema definition.
Schema element	Element declaration used as the signature if the signature source is from an XML schema definition.
Signature	The source for the input/output signature for the operation. Click **Modify Signature** to change the signature source. You can only change the operation signature source for a provider web service descriptor created from an existing IS service. You can use an element declaration in an external XML schema definition or an IS document type.
Addressing action	URI identifying the addressing action. The value for **Addressing action** cannot contain spaces or new line characters. When a WSDL is consumed for creating a consumer web service descriptor or a WSDL first provider web service descriptor and if the WSDL contains the addressing action attribute, **Addressing action** property will take the value of the addressing action attribute in WSDL. In case of a provider web service descriptor, the action specified in the binder for output or fault will be the addressing action in the SOAP response or fault respectively. If no action is specified in the binder, addressing action for the SOAP response is generated at runtime based on the Default Action Pattern for WSDL 1.1. For more information about the structure of the default generated action, see the WSDL 1.1 specification. You cannot edit the addressing action in a consumer web service descriptor. You can edit the addressing action property for service first and WSDL first provider web service descriptors only if WS-Addressing is enabled by attaching an addressing policy to the web service descriptor. **Note:** If both addressing action property and SOAP Action attribute are provided, the URI of the SOAP Action must be the same as the one specified by the addressing action property. If the values for the SOAP Action and the addressing action property are different, Integration Server uses the SOAP Action value while generating the addressing action in the SOAP request.

Header Element Properties

In the Properties view, under **General**, you can view basic information about a header element in an operation's request or response.

Property	Description
Name	Displays the name of the Header element.
Document type	Displays the fully qualified name of the IS Document type that defines the Header element.
Must understand	Indicates whether the Header element must be understood by the SOAP node in order for the message to be processed. "must Understand" is a SOAPHeader attribute. If mustUnderstand is set and the SOAP Node Receives a Header that it does not understand, it must *not* process the SOAP Request and the SOAP Node must return a "Not Understood" Fault. If this attribute is set to **True** for a consumer web service descriptor, the Must Understand attribute of the Header is set to true in the request. **Note:** The Must understand property can be edited for a header in a consumer web service descriptor or WSDL first provider web service descriptor only.
Role	URI naming the Actor (for SOAP 1.1) or Role (for SOAP 1.2) at which this header element is targeted. A Header is "targeted" at a SOAP Node if the node is acting in the role specified on that Header. The possible values are defined by the SOAP Specification. **Note:** The Role property can be edited for a header in a consumer web service descriptor or WSDL first provider web service descriptor only.

Fault Element Properties

In the Properties view, under **General**, you can view basic information about a fault element in an operation's response.

Property	Description
Name	Displays the name of the Fault element.
Document type	Displays the fully qualified name of the IS Document type defining the Fault element.

Property	Description
Addressing action	URI identifying the addressing action. The value for **Addressing action** cannot contain spaces or new line characters. When a WSDL is consumed for creating a consumer web service descriptor or a WSDL first provider web service descriptor and if the WSDL contains the addressing action attribute, **Addressing action** property will take the value of the addressing action attribute in WSDL. In case of a provider web service descriptor, the action specified in the binder for output or fault will be the addressing action in the SOAP response or fault respectively. If no action is specified in the binder, addressing action for the SOAP response is generated at runtime based on the Default Action Pattern for WSDL 1.1. For more information about the structure of the default generated action, see http://www.w3.org/TR/ws-addr-metadata/#defactionwsdl11. You cannot edit the addressing action in a consumer web service descriptor. You can edit the addressing action property for service first and WSDL first provider web service descriptors only if WS-Addressing is enabled by attaching an addressing policy to the web service descriptor. **Note:** If both addressing action property and SOAP Action attribute are provided, the URI of the SOAP Action must be the same as the one specified by the addressing action property. If the values for the SOAP Action and the addressing action property are different, Integration Server uses the SOAP Action value while generating the addressing action in the SOAP request.

Web Service Descriptor Binder Properties

In Properties view, you can view and set basic information about the binder for a web service descriptor when you select a binder in Binders tab in the web services descriptor editor. If the binder specifies SOAP over JMS as the binding style, you can view information included in the JMS binding and view an set information for the JMS message.

General Properties for Binders

When a binder is selected on the Binders tab, the Properties view displays general properties for the binder.

Property	Description
Binder name	Name of the binder.
Port address	Endpoint address associated with this web service, that is, the network address at which the web service can be invoked.

Property	Description
	For a consumer web service descriptor, this value is determined by the location attribute in the soap:address element (which is contained within the soap:port element of the service element). For a WSDL first provider web service descriptor, the **Port address** is empty. For a service first web service descriptor, you can edit the **Port address** for a binder that uses HTTP or HTTPS as the **Transport**. For a web service descriptor that uses the JMS transport, the **Port address** displays the initial part of the JMS URI, specifically "jms":<lookup var>:<dest>?targetService. Integration Server displays additional information that is part of the JMS URI in the JMS Settings and JMS Message Details properties. The **Port address** value is display-only when **Transport** is JMS, the binder is in a consumer web service descriptor, or the binder is in a WSDL first provider web service descriptor. **Note:** The contents of **Port address** might be used or overwritten when building the consumer and provider endpoint URLs for more information about how Integration Server constructs endpoint URLs, see the *Web Services Developer's Guide*.
Port alias	Endpoint alias name associated with this web service. The endpoint alias name will be used for this binder when generating a WSDL for a provider or when executing a web service connector for a consumer. The actual endpoint value is looked up at run time in both cases. New aliases can be defined from the Integration Server, using **Settings > Web Services**. For a provider web service and a binder with a protocol of HTTP or HTTPS, you can assign the default provider endpoint alias to the binder. Select DEFAULT(*aliasName*) if you want to use the information in the default provider web service endpoint alias. If the **Alias** list includes a blank row, Integration Server does not have a default provider web service endpoint alias for the protocol.
Port name	Name of the port associated with the web service, as defined by the WSDL; an aggregate of a binding and a network address.
Directive	The SOAP processor for which the web service will be a target. The drop-down menu lists all registered SOAP processors on the Integration Server to which you are currently connected.
Binding name	Name of the WSDL binding element.
Porttype name	Name of the portType associated with the WSDL binding element.

Property	Description
Binding type	Type of binding. This will be one of the following: ■ SOAP over HTTP ■ SOAP over JMS **Binding type** is display-only. The **Transport** value determines the **Binding type** value.
Transport	Transport mechanism used to invoke the web service, specify: ■ HTTP ■ HTTPS ■ JMS The transport you select must match the type of requests accepted by the port you selected.
SOAP version	Version of the SOAP message protocol to be used; either SOAP 1.1 or SOAP 1.2.
SOAP binding style	The style of the SOAP binding and its operations; either Document or RPC (Remote Procedure Call).
SOAP binding use	The usage attribute of the SOAP binding and its operations; either literal or encoded
SOAP binding transport	The transport protocol used by this SOAP Binding.
SOAP action	SOAP action associated with the operations in the binder. Click in the **Value** column to display the SOAP action string associated with each operation in the binder. For a service first provider web service descriptor, you can modify the SOAP action for an operation in a binder. The SOAP action value must be unique within the web service descriptor. The SOAP action cannot be edited for a WSDL first provider web service descriptor or a consumer web service descriptor.
Response endpoint address template	The address template that you can use as ReplyTo or FaultTo address to make the consumer web service descriptor process responses asynchronously. This property displays the following address format: ■ HTTP binder: **http://<server>:<port>/ws/*wsdName/portName*** ■ HTTPS binder: **https://<server>:<port>/ws/*wsdName/portName***

Property	Description
	■ JMS binder: **jms:<topic/queue/jndi>:<destinationName>?targetService=soapjms/*wsdName*/*portName***
	Where, ***wsdName*** is the web service descriptor name and ***portName*** is the name of the port associated with the web service, as defined by the WSDL; an aggregate of a binding and a network address. You must specify this address as the value for ReplyTo and/or FaultTo address in the *messageAddressingProperties* parameter of the corresponding web service connector to use this consumer web service descriptor to process responses asynchronously by invoking the callback response services. You must replace the placeholders <server> and <port> or <topic/queue/jndi> and <destinationName> with appropriate values depending on the transport mechanism used to invoke the web service.
Use CSQ	Indicates whether Integration Server places the request message in the client side queue if the JMS provider is not available at the time the message is sent. **Set to...** — **To...** **True** — Specify that Integration Server writes messages to the client side queue if the JMS provider is not available at the time the request message is sent by the web service connector. **False** — Specify that Integration Server throws an ISRuntimeException if the JMS provider is not available at the time the web service connector executes. This is the default. **Note:** This property applies to consumer web service descriptors only. For more information about using the client side queue when sending web service requests using SOAP over JMS, see"Configuring Use of the Client Side Queue" on page 798.

JMS Settings Properties for a Binder

If a binder specifies SOAP over JMS as the binding type, Designer displays JMS Settings properties for the binder. These properties specifies how to connect to the JMS provider and specify the destination to which messages are sent.

For a provider web service descriptor, the web service endpoint alias assigned to the binder determines the values of the JMS Settings properties.

For a consumer web service descriptor, the web service endpoint alias assigned to the binder and the WSDL used to create the web service descriptor determine the values of the JMS Settings properties. If the web service endpoint alias and the WSDL specify a value for the same JMS Settings property, the value in the web service endpoint alias takes precedence.

The properties displayed under JMS Settings vary depending on the value of the **Variant identifier** property.

Property	Description
Variant identifier	Specifies how a destination name is looked up. The Variant identifier corresponds to the jms-variant syntax in the JMS URI Schema. The **Variant identifier** will be one of the following:

Value	Description
jndi	Indicates that JNDI is used to lookup the administered objects (connection factory and destination) needed to send the JMS message.
queue	Indicates that destination is a queue and indicates that the client uses the native webMethods API to connect directly to webMethods Broker.
topic	Indicates that destination is a topic and indicates that the client uses the native webMethods API to connect directly to webMethods Broker.

Destination	If the **Variant identifier** is jndi, specifies the JNDI provider lookup name for the destination to which messages are sent on the JMS provider. If the **Variant identifier** is queue or topic, specifies the name of the destination to which messages are sent.
JMS connection alias	Name of the JMS connection alias used to connect to the JMS provider. Designer displays this property only when the **Variant identifier** is "queue" or "topic".
JNDI connection factory name	JNDI provider lookup name for the connection factory used to create a connection to the JMS provider. Designer displays this property only when the **Variant identifier** is "jndi".
JNDI initial context factory	Java class name of the InitialContextFactory for the JNDI provider. Designer displays this property only when the **Variant identifier** is "jndi".
JNDI URL	Location of the registry when the registry is being used as the initial context. Designer displays this property only when the **Variant identifier** is "jndi".
Other properties	Any additional properties the JNDI provider requires for configuration. Designer displays this property only when the **Variant identifier** is "jndi".

JMS Message Details Properties for a Binder

If a binder specifies SOAP over JMS as the binding type, Designer displays JMS Message Details properties for the binder. These properties display the JMS message header information for the request message, such as delivery mode, time to live, and the destination for replies. The JMS Message Details properties are read-only.

For a provider web service descriptor, the web service endpoint alias assigned to the binder's port alias determines the values of the properties under JMS Message Details. A blank property indicates that the web service endpoint alias does not specify a value for the property. For example, if the web service endpoint alias does not specify a delivery mode, the **Delivery mode** property under JMS Message Details will be blank too.

For a consumer web services descriptor, the binding information in the WSDL document used to create the consumer web service descriptor determines the values of the JMS Message Details properties. If the WSDL document does not contain information that Integration Server uses to populate a property for the corresponding binding, the property will be blank. For example, if the WSDL does not contain the soapjms:timetolive element, the **Time to live** property will be blank in the binder.

Property	Description
Delivery mode	The message delivery mode for the request message. This is the delivery mode that web service clients must specify in the JMS message that serves as the request message for the web service.

Value	Description
PERSISTENT	Indicate the request message should be persistent. The message will not be lost if the JMS provider fails.
NON_PERSISTENT	Indicate the request message is not persistent. The message might be lost if the JMS provider fails.

Time to live	The number of milliseconds that can elapse before the request message expires on the JMS provider. A value of 0 indicates that the message does not expire.
Priority	Specifies the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.
Reply to name	Name or lookup name of the destination to which the web service sends a response (reply) message.
Reply to type	Type of destination to which the web service sends the response (reply) message.

Value	Description
QUEUE	Indicates that the web service sends the response message to a particular queue.
TOPIC	Indicates that the web service sends the request message to a particular topic.

Property	Description
	Note: The **Reply to type** property is only applicable when the **Variant identifier** is “queue” or “topic”.

Web Service Descriptor Header Handler Properties

You can view basic information about the header handlers for the web service descriptor in the Properties view.

Property	Description
Handler name	Name of the header handler.
Class name	The Java class name of the web service handler based on JAX-RPC that acts as the header handler.
Policy type	The policy type associated with the header handler. Policy files used with this header handler must be of this type.
Policy name	Specifies the name of the policy assigned to this header handler. The policy name is obtained from the ID attribute in the policy file. At run time, the assigned policy can be overridden by the value of the **Effective policy name** property.
Effective policy name	Specifies the name of the policy used with this header handler at run time. The effective policy overrides the policy assigned in the **Policy name** property.

webMethods Messaging Trigger Properties

Use the Properties view to display information about webMethods Messaging Triggers, specify error handling, specify message processing, configure exactly-once processing, and assign permissions.

To view properties for a webMethods Messaging Trigger, double-click the webMethods Messaging Trigger in the Package Navigator view of Designer.

General Properties for webMethods Messaging Triggers

In the Properties view, under **General**, you can enable/disable the webMethods Messaging Trigger, configure join expiration, enable priority message handling, and assign permissions to the trigger.

Property	Description
Enabled	Specifies whether the webMethods Messaging Trigger is enabled or disabled.

Property	Description	
	Select...	**To...**
	True	Enable the webMethods Messaging Trigger. An enabled webMethods Messaging Trigger contains one or more valid conditions.
	False	Disable the webMethods Messaging Trigger. A disabled webMethods Messaging Trigger can contain valid or invalid conditions. A disabled webMethods Messaging Trigger does not have any subscriptions registered on the Integration Server or the messaging provider.
Join expires	Indicates whether the join expires after the time period specified in **Expire after**.	
	Select...	**To...**
	True	Indicate Integration Server should stop waiting for the remaining documents in a join condition after the time specified in **Expire after** elapses.
	False	Indicate that the join should not expire. That is, Integration Server should wait indefinitely for the remaining documents in a join condition.
	Note:webMethods Messaging Triggers that receive documents from Universal Messaging do not use joins. Designer ignores the value of the **Join expires** property if the trigger subscribes to publishable document types that can be published to Universal Messaging.	
Expire after	Specifies how long Integration Server waits for the remaining documents in the join condition. The default join time-out period is 1 day.	
	Note:webMethods Messaging Triggers that receive documents from Universal Messaging do not use joins. Designer ignores the value of the **Expire after** property if the trigger subscribes to publishable document types that can be published to Universal Messaging.	
Priority enabled	Specifies whether priority messaging is enabled or disabled for the webMethods Messaging Trigger. This property applies to webMethods Messaging Triggers that receive documents from Broker only. webMethods Messaging Triggers that receive documents from Universal Messaging always receive higher priority documents in an expedited fashion. Additionally, priority messaging does not apply to locally published documents received by the webMethods Messaging Trigger. At run time,	

Property	Description
	Integration Server ignores the value of the **Priority enabled** property if the trigger receives a locally published document.

Select...	To...
True	Enable priority messaging for the webMethods Messaging Trigger.
	Note: If priority messaging is enabled on an existing trigger, the Broker connection is dropped and the client is deleted. The client gets recreated when the Broker connection is re-established. This may result in loss of documents for that trigger.
False	Disable priority messaging for the trigger.

Execution user

Specifies the name of the user account whose credentials Integration Server uses to execute a service associated with the webMethods Messaging Trigger. You can specify a locally defined user account or a user account defined in a central or external directory.

Note:
The **Execution user** property only applies to webMethods Messaging Triggers that receive documents from Universal Messaging. The publishable document type to which a trigger subscribes determine the messaging provider from which the trigger receives messages. At run time, Integration Server ignores the value of the **Execution user** property if a webMethods Messaging Trigger receives locally published documents or documents from Broker

Permissions

Click ... to assign or view ACL permissions to a webMethods Messaging Trigger.

Reuse

Specifies whether this element can be dragged from the CentraSite Registry Explorer view to a BPM process or CAF project.

When this property is set to public, you can drag the asset to a BPM process or CAF project.

When this property is set to private (the default), you cannot drag the asset to a BPM process or CAF project.

All published assets are available for Impact Analysis, whether they are public or private.

Although changing the public/private status will immediately change whether or not you can drag an element to a BPM process or CAF project, the element's

Property	Description
	status in CentraSite will not change until the next publication of assets to CentraSite.

Trigger Queue Properties

In the Properties view, under **Trigger queue**, you can specify the capacity and refill levels of the trigger queue on Integration Server. You can also specify how many messages Integration Server should acknowledge at one time.

Property	Description
Capacity	Specifies the maximum number of unprocessed documents that can exist for this trigger in the trigger document store. (Each trigger has its own queue in the trigger document store on Integration Server.) The default is 10 documents.
Refill level	Specifies the number of unprocessed documents that must remain in this trigger queue before the Integration Server retrieves more documents for the trigger. The default is 4 documents. **Note:** The **Refill level** does not apply to webMethods Messaging Triggers that receive documents from Universal Messaging. At run time, Integration Server ignores the value of the **Refill level** property if a webMethods Messaging Trigger receives documents from Universal Messaging.
Acknowledgement queue size	Specifies the maximum number of pending document acknowledgements for this trigger. A server thread places acknowledgements in the acknowledgement queue after it finishes executing the trigger service. Acknowledgements collect in the queue until the server returns them as a group to the sending resource. If the acknowledgement queue fills to capacity, the server blocks any server threads that attempt to add an acknowledgement to the queue. The blocked threads resume execution only after Integration Server empties the queue by returning the acknowledgements. The default is 1 acknowledgement.

Message Processing Properties

In the Properties view, under **Message processing**, you can specify whether the trigger processes messages serially or concurrently.

Property	Description
Processing mode	Specifies how Integration Server processes the documents in the trigger queue.

Property	Description	
	Select...	**To...**
	Serial	Specify that Integration Server should process documents in the trigger queue one after the other, in the same order that they were received.
	Concurrent	Specifies that Integration Server should process as many documents in the trigger queue as it can at one time. The maximum number of documents the server can process at one time is determined by the **Max execution threads** property.
Max execution threads	Specifies the maximum number of server threads that can process documents for this trigger concurrently. Integration Server uses one server thread to process each document in the trigger queue. The default is 1 server thread.	

Fatal Error Handling Properties

In the Properties view, under **Fatal error handling**, you can specify how Integration Server responds when a trigger service ends because of a fatal error.

Property	Description	
Suspend on error	Specifies that Integration Server suspends document retrieval and document processing when an exception occurs during trigger service execution. This property is only available for serial triggers.	
	Select...	**To...**
	True	Suspend document processing and document retrieval for the trigger when a trigger service ends with an error.
	False	Indicate that document processing and document retrieval should not be suspended if a trigger service ends with an error.

Transient Error Handling Properties

In the Properties view, under **Transient error handling**, you can specify how Integration Server responds when a trigger service ends because of a transient error.

Property	Description
Retry until	Specifies the maximum number of times that Integration Server will attempt to execute the trigger service if an ISRuntimeException occurs during the trigger service execution. An ISRuntimeException occurs when the trigger service catches a transient error, wraps the error, and re-throws it as an exception. (A *transient error* is an error that arises from a condition that might

Property	Description
	be resolved quickly, such as the unavailability of a resource due to network issues or failure to connect to a database.) **Select...** / **To...** **Max attempts reached** — Indicate that Integration Server re-executes the trigger service up to the number of times specified in the **Max attempts** property. The server stops re-executing the trigger service when the service succeeds or when the server reaches the maximum number of retries. **Successful** — Indicate that Integration Server re-executes the trigger service until the service executes to completion.
Max retry attempts	Specifies the maximum number of times the Integration Server should re-execute the trigger service if an ISRuntimeException occurs during service execution. The default is 0 attempts, which indicates that Integration Server does not retry the trigger service.
Retry interval	Specifies the length of time Integration Server waits between attempts to execute the trigger service. The default is 10 seconds.
On retry failure	Indicates how Integration Server handles a retry failure for a trigger. A retry failure occurs when Integration Server reaches the maximum number of retry attempts and the trigger service still fails because of an ISRuntimeException. This property also determines how Integration Server handles a transient error that occurs during trigger preprocessing. **Select...** / **To...** **Throw exception** — Indicate that Integration Server throws a service exception when the last allowed retry attempt ends because of an ISRuntimeException. This is the default. **Suspend and retry later** — Indicate that Integration Server suspends the trigger when the last allowed retry attempt ends because of a run-time exception. Integration Server retries the trigger service at a later time when the resources needed by the trigger service become available. When **On Retry failure** is set to **Suspend and retry later**, a transient error that occurs during trigger preprocessing causes Integration Server to suspend the trigger and resume it when the resources, specifically the document history database, are available.

Property	Description
Resource monitoring service	Specifies the service that Integration Server executes to determine whether the resources needed by the trigger are available and if the trigger can be resumed. Integration Server schedules a system task to execute the resource monitoring service when one of the following occurs: ■ The trigger service ends because of a retry failure and the **On retry failure** property is set to **Suspend and retry later**. ■ The document resolver service used for exactly-once processing ends because of a run-time exception and the watt.server.trigger.preprocess.suspendAndRetryOnError is set to true. **Note:** A resource monitoring service must use the pub.trigger:resourceMonitoringSpec as the service signature.

Exactly Once Properties

In the Properties view, under **Exactly once**, configure exactly-once processing for a trigger. Exactly-once processing ensures that a trigger processes a guaranteed document once and only once. Integration Server provides exactly-once processing by determining whether a document is a copy of one previously processed by the trigger. Designer provides three duplicate detection methods: redelivery count, document history, and a document resolver service.

Property	Description
Detect duplicates	Enables exactly-once processing for the trigger and instructs the server to check a document's redelivery count to determine whether the trigger has received the document before. **Select...** / **To...** **True** — Specify that exactly-once processing is provided for documents received by this trigger and instructs the server to check a document's redelivery count to determine whether the trigger received the document previously. The redelivery count indicates the number of times the routing resource has redelivered a document to the trigger. **False** — Specify that exactly-once processing is not provided for documents received by this trigger.
Use history	Indicates whether a document history database will be maintained and used to determine whether a document is a duplicate. **Select...** / **To...**

Property	Description	
	True	Indicate that Integration Server maintains a history of documents processed by the trigger. When the trigger receives a document, Integration Server compares the document's universally unique identifier (UUID) to the UUIDs of documents processed by the trigger. If there is a match, the Integration Server either determines the second document is a duplicate and discards it or, if the first document has not finished processing, marks the second document's status as In Doubt.
	False	Indicate that Integration Server does not maintain a document history database. The Integration Server will not use document history to determine whether a document is a duplicate of one already processed by the trigger.
	Note: To perform duplicate detection using a document history database, the audit subsystem must be stored in a relational database and Integration Server Administrator must define a JDBC connection pool for the Integration Server to use to connect to the document history database.	
History time to live	Specifies the length of time the document history database maintains an entry for a document processed by the trigger. During this time period, Integration Server discards any documents with the same universally unique identifier (UUID) as an existing document history entry for the trigger. When a document history entry expires, Integration Server removes it from the document history database. If the trigger subsequently receives a document with same UUID as the expired and removed entry, the server considers the copy to be new because the entry for the previous document has been removed from the database.	
Document resolver service	Specifies the service that you created to determine whether message's status is New, Duplicate, or In Doubt. Click ... to select a service from a list. The document resolver service must use the pub.publish:documentResolverSpec to define the service signature.	

WebSocket Endpoint Properties

General Properties for WebSocket Endpoint

In the Properties view, under **General**, you can view and assign properties to a WebSocket Endpoint. General properties for the element you select in the Package Navigator appear in the Properties view. From the Properties view you can also view and set permissions for the element.

Property	Description
Name	Displays the name of the WebSocket endpoint.
Permissions	Click to assign ACL permissions to a service.
Behavior	Displays whether the WebSocket endpoint is for client or for a server.
URI	Displays the location of the source URI used to create the WebSocket endpoint.

52 webMethods Flow Steps

- BRANCH 1070
- EXIT 1073
- INVOKE 1074
- LOOP 1075
- MAP 1077
- REPEAT 1078
- SEQUENCE 1080

A flow step is a basic unit of work (expressed in the webMethods flow language) that webMethods Integration Server interprets and executes at run time. The webMethods flow language provides the following flow steps that invoke services and flow steps that let you edit data in the pipeline:

- BRANCH
- EXIT
- INVOKE
- LOOP
- MAP
- REPEAT
- SEQUENCE

BRANCH

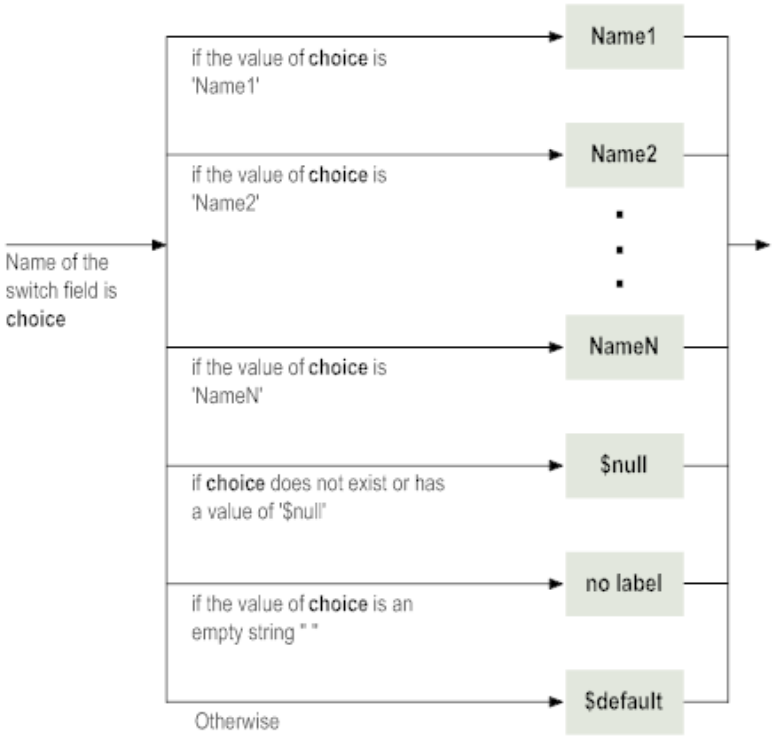
The BRANCH step selects and executes a child step based on the value of one or more variables in the pipeline. You indicate the variables you want to branch on by specifying a switch value or by writing an expression that includes the variables.

Branching on a Switch Value

When you branch on a switch value, you specify the switch variable in the **Switch** property of the BRANCH step. In the **Label** property for each child step, you specify the value of the switch variable that will cause that child step to execute. At run time, the BRANCH flow step executes the child step that has the same label as the value of the **Switch** property.

If you want to execute a child step when the value of the **Switch** property is an empty string, leave the **Label** property of the child step blank. If you want to execute a child step when the **Switch** property is a null or unmatched string, set the **Label** of the child step to **$null** or **$default**.

BRANCH flow step using a switch

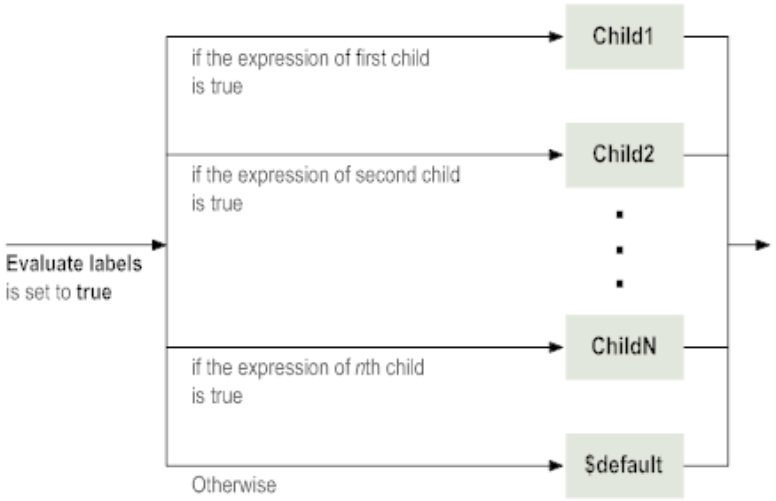


Branching on Expressions

When you branch on expressions, you set the **Evaluate labels** property of the BRANCH step to true. In the **Label** property for each child step, you write an expression that includes one or more variables. At run time, the BRANCH step executes the first child step with an expression that evaluates to true.

If you want to specify a child step to execute when none of the expressions are true, set the label of the child step to **$default**.

BRANCH step using expressions



BRANCH Properties

The BRANCH step has the following properties.

Property	Description
Comments	Optional. Specifies a descriptive comment for the step.
Scope	Optional. Specifies the name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Label	Optional. (Required if you are using this BRANCH step as a target for another BRANCH or EXIT step.) Specifies a name for this instance of the BRANCH step, or a null, unmatched, or empty string ($null, $default, blank).
Switch	Specifies the String field that the BRANCH step uses to determine which child flow step to execute. The BRANCH step executes the child flow step whose label matches the value of the field specified in the **Switch** property. Do not specify a value if you set the **Evaluate labels** property to **True**.

Property	Description
Evaluate labels	Specifies whether or not you want the server to evaluate labels of child steps as conditional expressions. When you branch on expressions, you enter expressions in the **Label** property for the children of the BRANCH step. At run time, the server executes the first child step whose label evaluates to **True**. To branch on expressions, select **True**. To branch on the **Switch** value, select **False**.

Conditions that Will Cause a BRANCH Step to Fail

- The switch field is not in the pipeline and the BRANCH step does not contain a default child step or a child step to handle null values.
- The matching child step fails.
- The BRANCH step does not complete before the time-out period expires.

EXIT

The EXIT step exits the entire flow service or a single flow step. Specifically, it may exit from the nearest ancestor loop step, a specified ancestor step, the parent step, or the entire flow service.

The EXIT step can throw an exception if the exit is considered a failure. When an exception is thrown, user-specified error message text is displayed by typing it directly or by assigning it to a variable in the pipeline.

EXIT Properties

The EXIT step has the following properties.

Property	Description
Comments	Optional. Specifies a descriptive comment for the step.
Label	Optional. (Required if you are using this EXIT step as a target for a BRANCH step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).
Exit from	Required. Specifies the flow step or service from which you want to exit.

Specify this value…	To exit the…
$parent	Parent flow step, regardless of the type of step.
$loop	Nearest parent LOOP or REPEAT step.
$flow	Entire flow.
label	Nearest ancestor step that has a label that matches this value.

Property	Description
	Note: If the label you specify does not match the label of an ancestor flow step, the flow will exit with an exception.
Signal	Required. Specifies whether the exit is considered a success or a failure. A **SUCCESS** condition exits the flow service or step. A **FAILURE** condition exits the flow service or step and throws an exception. The text of the exception message is contained in the **Failure message** property.
Failure message	Optional. Specifies the text of the exception message that is displayed when **Signal** is set to **FAILURE**. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%mymessage%`. The variable you specify must be a String.

Examples of When to Use an EXIT Step

- Exit an entire flow service from within a series of deeply nested steps.
- Throw an exception when you exit a flow service or a flow step without having to write a Java service to call Service.throwError( ).
- Exit a LOOP or REPEAT flow step without throwing an exception.

INVOKE

The INVOKE flow step invokes another service. You can use it to invoke any type of service, including another flow service.

INVOKE Properties

The INVOKE step has the following properties.

Property	Description
Comments	Optional. Specifies a descriptive comment for the step.
Label	Optional. Specifies the name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service.

Property	Description
	If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Service	Required. Specifies the fully qualified name of the service to invoke.
Validate input	Optional. Specifies whether the server validates the input to the service against the service input signature. If you want the input to be validated, select **True**. If you do not want the input to be validated, select **False**.
Validate output	Optional. Specifies whether the server validates the output of the service against the service output signature. If you want the output to be validated, select **True**. If you do not want the output to be validated, select **False**.

Conditions that Will Cause an INVOKE Step to Fail

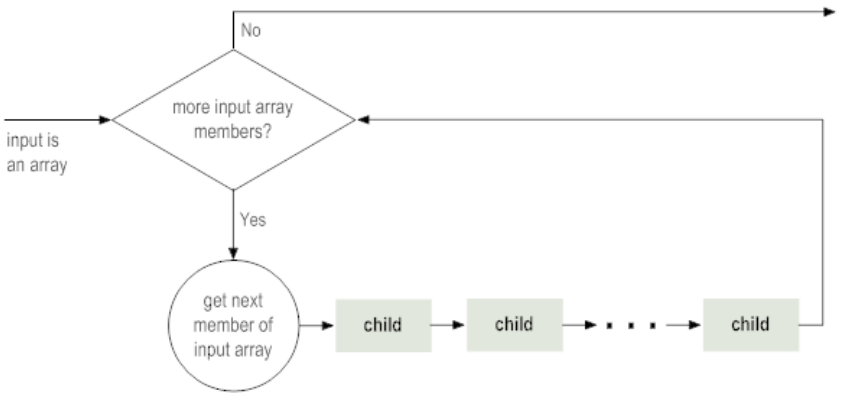
- The service that is invoked fails.
- The specified service does not exist.
- The specified service is disabled.

LOOP

The LOOP step takes as input an array variable that is in the pipeline. It loops over the members of an input array, executing its child steps each time through the loop. For example, if you have a service that takes a string as input and a string list in the pipeline, use the LOOP step to invoke the service one time for each string in the string list.

You identify a single array variable to use as input when you set the properties for the LOOP step. You can also designate a single variable for output. The LOOP step collects an output value each time it runs through the loop and creates an output array that contains the collected output values. If you want to collect more than one variable, specify a document that contains the fields you want to collect for the output variable.

The LOOP step



LOOP Properties

The LOOP step has the following properties.

Property	Description
Comments	Optional. Specifies a descriptive comment for the step.
Scope	Optional. Specifies the name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Label	Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).
Input array	Required. Specifies the input array over which to loop. You must specify a variable in the pipeline that is an array data type (that is, String list, String table, document list, or Object list).
Output array	Optional. Specifies the name of the field in which the server places output data for an iteration of the loop. The server collects the output from the iterations into an array field with the same name. You do not need to specify this property if the loop does not produce output values.

Conditions that Will Cause a LOOP Step to Fail

- The pipeline does not contain the input array.
- The input field is not an array field.
- A child step of the LOOP step fails during any iteration of the loop.
- The LOOP step does not complete before the time-out period expires.

MAP

The MAP step adjusts the pipeline at any point in a flow. It makes pipeline modifications that are independent of an INVOKE step.

Within the MAP step, you can:

- Link (copy) the value of a pipeline input field to a new or existing pipeline output field.
- Drop an existing pipeline input field. (Keep in mind that once you drop a field from the pipeline, it is no longer available to subsequent services in the flow.)
- Assign a value to a pipeline output field.
- Perform document-to-document mapping in a single view by inserting transformers.

MAP Properties

The MAP step has the following properties.

Property	Description
Comments	Optional. Specifies a descriptive comment for this step.
Scope	Optional. Specifies the name of a document (IData) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.

Property	Description
Label	Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).

Example of When to Use a MAP Step

- You want to assign an initial set of input values in a flow service (that is, to initialize variables). You insert the MAP step at the beginning of the flow, and then use the **Set Value** modifier to assign values to the appropriate variables in **Pipeline Out**.
- You want to map a document from one format to another (for example, cXML to XML). Insert transformers into the MAP step to perform the needed data transformations.

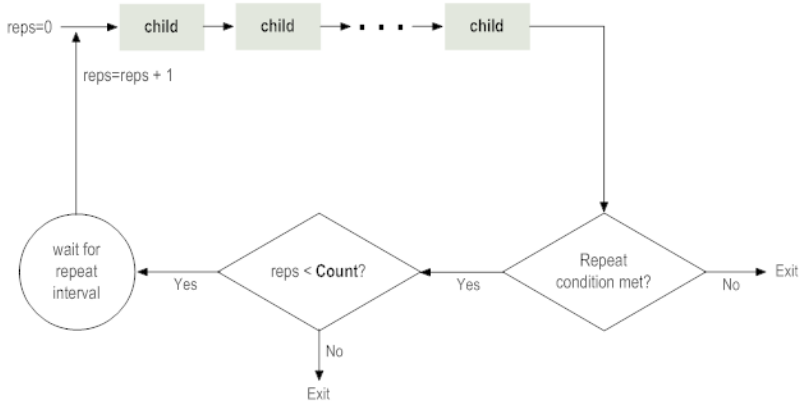
REPEAT

The REPEAT step repeatedly executes its child steps up to a maximum number of times that you specify. It determines whether to re-execute the child steps based on a **Repeat on** condition. You can set the repeat condition to one of the following:

- Repeat if any one of the child steps fails.
- Repeat if all of the elements succeed.

You can also specify a time period that you want the REPEAT flow step to wait before it re-executes its child steps.

The REPEAT step



REPEAT Properties

The REPEAT step has the following properties.

Property	Description
Comments	Optional. Specifies a descriptive comment for this step.
Scope	Optional. Specifies the name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Label	Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).
Count	Required. Specifies the maximum number of times the server re-executes the child steps in the REPEAT step. Set **Count** to `0` (zero) to instruct the server that the child steps should not be re-executed. Set **Count** to a value greater than zero to instruct the server to re-execute the child steps up to a specified number of times. Set **Count** to `-1` to instruct the server to re-execute the child steps as long as the specified **Repeat on** condition is true. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%servicecount%`. The variable you specify must be a String.
Repeat interval	Optional. Specifies the number of seconds the server waits before re-executing the child steps. Specify `0` (zero) to re-execute the child steps without a delay. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%waittime%`. The variable you specify must be a String.
Repeat on	Required. Specifies when the server re-executes the REPEAT child steps. Select **SUCCESS** to re-execute the child steps when the all the child steps complete successfully. Select **FAILURE** to re-execute the child steps when any one of the child steps fails.

When Does REPEAT Fail?

The following conditions cause the REPEAT step to fail:

If "Repeat on" is set to…	The REPEAT step fails if…
SUCCESS	A child within the REPEAT block fails.
FAILURE	The **Count** limit is reached before its children execute successfully.

If the REPEAT step is a child of another step, the failure is propagated to its parent.

Examples of When to Use a REPEAT Step

- **"Repeat on" property is set to FAILURE.** Use when a service accesses a remote server and you want the service to retry if the server is busy. Make the service that accesses the remote server a child element of a REPEAT flow step, and then set the **Repeat on** property to **FAILURE**. If the service attempts to access the website and it fails, the REPEAT flow step attempts to retry the service again. You also set a **Repeat interval** that causes the REPEAT flow condition to wait a period of time before invoking the service again.
- **"Repeat on" property is set to SUCCESS.** Use in a web-automation service when you want to repeat a load and query step and a "Next Page" button exists in the current document, indicating that there are additional pages to be processed. End the REPEAT flow step when the query step fails to retrieve a "Next Page" button in the current document.

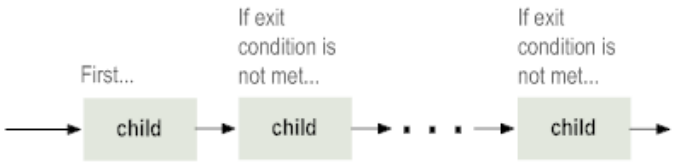
SEQUENCE

The SEQUENCE step forms a collection of child steps that execute sequentially. This is useful when you want to group a set of steps as a target for a BRANCH step.

You can set an exit condition that indicates whether the SEQUENCE should exit prematurely and, if so, under what condition. Specify one of the following exit conditions:

- **Exit the SEQUENCE when a child step fails.**Use this condition when you want to ensure that all child steps are completed successfully. If any child step fails, the SEQUENCE ends prematurely and the sequence fails.
- **Exit the SEQUENCE when a child step succeeds.** Use this condition when you want to define a set of alternative services, so that if one fails, another is attempted. If a child step succeeds, the SEQUENCE ends prematurely and the sequence succeeds.
- **Exit the SEQUENCE after executing all child steps.** Use this condition when you want to execute all of the child steps regardless of their outcome. The SEQUENCE does not end prematurely.

The SEQUENCE step



SEQUENCE Properties

The SEQUENCE step has the following properties.

Property	Description
Comments	Optional. Specifies a descriptive comment for this step.
Scope	Optional. Specifies the name of a document (IData object) in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this property blank.
Timeout	Optional. Specifies the maximum number of seconds that this step should run. If this time elapses before the step completes, Integration Server issues a FlowTimeoutException and execution continues with the next step in the service. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, `%expiration%`. The variable you specify must be a String. If you do not need to specify a time-out period, leave **Timeout** blank. For more information about how Integration Server handles flow step timeouts, refer to the description of the watt.server.threadKill.timeout.enabled configuration parameter in *webMethods Integration Server Administrator's Guide*.
Label	Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).
Exit on	Required. Specifies when to exit the SEQUENCE step.

Specify this value…	To...
FAILURE	Exit the SEQUENCE when a child step fails. Execution continues with the next flow step in the flow service. The SEQUENCE step executes its child steps until either one fails or until it executes all its child steps. This is the default.
SUCCESS	Exit the SEQUENCE when a child step executes successfully or after all child steps fail. Execution continues with the next flow step in the flow service.

Property	Description
	The SEQUENCE step executes its child steps until either one succeeds or until it executes all its child steps and all of the child steps fail. **Note:** Successful execution of a MAP step within a SEQUENCE step, including successful execution of any transformers, does not cause the containing SEQUENCE to exit when **Exit on** is set to **SUCCESS.**
DONE	Exit the sequence after all child steps execute. The SEQUENCE step executes *all* of its child steps regardless of whether they succeed or fail.

Conditions that Will Cause the SEQUENCE Step to Fail

This section describes the conditions that cause failure based on the exit condition for the sequence.

If **Exit on** is set to **FAILURE**, conditions that will cause a failure include:

- One of the child steps fails.
- The SEQUENCE step does not complete before the time-out period expires.

If **Exit on** is set to **SUCCESS**, conditions that will cause a failure include:

- All the child steps fail.
- The SEQUENCE step does not complete before the time-out period expires.

If **Exit on** is set to **DONE**, conditions that will cause a failure include:

- The SEQUENCE step does not complete before the time-out period expires.

53 Data Types

- Data Types in IData Objects 1084
- Java Classes for Objects 1085
- How Designer Supports Tables 1087

Designer supports several data types for use in services. Each data type supported by Designer corresponds to a Java data type and has an associated icon. Designer applies different Java classes and displays different icons depending on whether the data type is associated with:

- An element in an IData object
- An Object or Object list to which you have applied a Java class

Note: Designer does not provide a separate data type for tables.

Data Types in IData Objects

Data is passed in and out of a service through an IData object. An IData object is the collection of name/value pairs on which a service operates. An IData object can contain any number of elements of any valid Java objects, including additional IData objects and IDataCodable objects.

Each element stored in an IData object corresponds to a data type. The following table identifies the data types supported by Designer.

Data Type	Icon	Description	Java Class
String		String of characters.	java.lang.String
String list		A one-dimensional String array.	java.lang.String[ ]
String table		A two-dimensional String array.	java.lang.String[ ][ ]
Document		A data structure that is a container for other variables. Documents can contain variables of any other data type. The contents of a document (IData object) are stored as key/value pairs where the variable name is the key.	com.wm.data.IData com.wm.util.Values For more information, see the *webMethods Integration Server Java API Reference*.
Document list		A one-dimensional array of IS document types (IData [ ]or Values [ ]).	com.wm.data.IData [ ] com.wm.util.Values [ ] com.wm.util.Table
Document reference		A document whose structure is defined by an IS document type.	Reference to an existing object which implements the com.wm.data.IData interface or a reference to an existing com.wm.util.Values object.
Document reference list		A document list whose structure is defined by an IS document type.	Reference to an existing object which implements the com.wm.data.IData interface or a reference to an existing com.wm.util.Values object.

Data Type	Icon	Description	Java Class
Object		A data type that does not fall into any of the data types described in the above rows, and is not declared to be one of the basic Java classes supported natively by Integration Server. This icon is used for Objects of unknown type.	Any subclass of java.lang.Object. **Example:** java.io.InputStream
Object list		An array of Objects of unknown type.	An array of any subclass of java.lang.Object. **Example:** java.io.InputStream[ ]

Note: Designer displays small symbols next to variable icons to indicate validation constraints. Designer uses ▫ to indicate an optional variable and the ‡ symbol to denote a variable with a content constraint. Designer also uses ▲ to indicate that the variable has a default value that can be overridden assigned to it and 🔒 to indicate that the variable has a null value that cannot be overridden assigned to it. A combination of the ▲ and 🔒 symbols next to a variable icon indicates that the variable has a fixed default value that is not null and cannot be overridden.

Java Classes for Objects

You can further describe the contents of an Object or Object list variable by applying a Java class to the variable. When you apply a supported Java class to an Object or Object list variable, Designer changes the icon for the variable. Applying Java classes to Objects and Object lists can provide the following benefits:

- Other developers can easily see the types your service expects as inputs and produces as output.
- Other developers can easily see the types contained in an IS document type.
- You can input values for the variable when running and debugging.
- You can assign values to variables in the pipeline using ⚙ on the Pipeline view toolbar.

Note:
When you input values for a constrained Object during debugging or when assigning a value in the pipeline, Designer validates the data to make sure it is of the correct type.

The following table identifies the Java classes you can apply to Objects and Object list variables in Designer.

Data Type	Icon	Description	Java Class
boolean		True or false.	java.lang.Boolean

Data Type	Icon	Description	Java Class
boolean list		A one-dimensional boolean array.	java.lang.Boolean[ ]
byte		Signed integer. The value must be greater than or equal to –128 but less than or equal to 127.	java.lang.Byte
byte [ ]		A one-dimensional byte array.	primitive type
byte list		A one-dimensional byte array.	java.lang.Byte[ ]
character		A single unicode character.	java.lang.Character
character list		A one-dimensional character array.	java.lang.Character[ ]
date		Date and time.	java.util.Date
date list		A one-dimensional date array.	java.util.Date[ ]
double		Double-precision floating point number.	java.lang.Double
double list		A one-dimensional double array.	java.lang.Double[ ]
float		Standard-precision floating point number.	java.lang.Float
float list		A one-dimensional float array.	java.lang.Float[ ]
integer		Signed integer. The value must be greater than or equal to -2147483648 but less than or equal to 2147483647.	java.lang.Integer
integer list		A one-dimensional integer array.	java.lang.Integer[ ]
long		Signed integer. The value must be greater than or equal to –9223372036854775808 but less than or equal to 9223372036854775807.	java.lang.Long
long list		A one-dimensional long array.	java.lang.Long[ ]
short		Signed integer. The value must be greater than or equal to -32768 but less than or equal to 32767.	java.lang.Short
short list		A one-dimensional short array.	java.lang.Short[ ]
XOPObject		A field in a SOAP message that is to be sent/received as a streamed MTOM attachment.	com.wm.util.XOPObject

Data Type	Icon	Description	Java Class
		Note: Integration Server only supports this Java wrapper type for web services.	

How Designer Supports Tables

With the exception of String table, Designer does not provide a separate data type for tables. However, tables can appear as document lists or Objects. Tables that are instances of com.wm.util.Table appear as document lists in Designer. These tables can be used as document lists in flow services. Services in the WmDB package use tables that are instances of wm.com.util.Table.

Tables can also be declared as Objects. Objects or user-defined table-like objects that do not implement the com.wm.util.pluggable.WMIDataList interface appear as Objects of unknown type in Designer.

54 Icons

- Package Navigator View Icons 1090
- UDDI Registry View Icons 1093
- Flat File Element Icons 1093
- Flow Step Icons 1094
- OData Service Icons 1095
- REST API Descriptor Icons 1096
- Schema Component Icons 1096

This topic describes the icons used to identify elements in the Service Development perspective.

Package Navigator View Icons

The following icons are used to represent elements in the Package Navigator view.

This icon...	Represents a...
	Server. You can have multiple server contexts displayed in Designer. The active server context is the one that is highlighted in Package Navigator view. To display the contents of the server, click the ⊞ symbol next to its name.
	Server. The Integration Server instance that is currently used as the as the (local development server.
	Server. The Integration Server instance that is currently not used as a local development server.
	Package. A package contains a set of services and related files, such as specifications, IS document types, and output templates. To display the contents of a package, click ⊞ next to its name. **Note:**Designer places Trading Networks document types in a package named "Trading Networks Documents".
	Folder.A folder contains related services and optional folders (called subfolders). To display the contents of a folder, click ⊞ next to its name.
	REST resource folder. A folder that contains the services that act as REST resources. To display the services for a REST resource, click ⊞ next to its name. Services can be named _get, _put, _post, _patch, _delete, or _default.
	Flow service. A flow service is a service written in the webMethods flow language.
	Java service. A Java service is a service written in Java.
	C service. A C service is a service written in C/C++.
	Cloud connector service. A cloud connector service works with webMethods CloudStreams to integrate an on-premise application with a SaaS application.
	.NET service. A .NET service is a service that calls methods imported from .NET assemblies (using the webMethods Microsoft Package).
	OData service. An OData service is a service that exposes and allows clients to access the data in an OData data model.
	IS document type. An IS document type contains a set of fields used to define the structure and type of data in a document.
	Publishable document type. A publishable document type is an IS document type with specific publishing properties. Instances of publishable document types can

This icon...	Represents a...
	be published and subscribed to. Publishable document types can be used anywhere an IS document type is needed.
	REST API descriptor. A REST API descriptor is a collection of REST resources and meta data, including how to access the resources and what parameters are expected and returned, that describe a REST API.
	XML document type. An XML document type corresponds to a global element declaration, global attribute declaration, or global complex type definition in an XML schema definition.
	XML field. An XML field corresponds to a global element declaration with simple content as defined in an XML schema definition.
	Specification. A specification is a formal description of a service's inputs and outputs.
	IS schema. An IS schema is the blueprint or model document against which you validate an XML document. The schema defines what can and cannot be contained in the XML documents it validates.
	webMethods Messaging Trigger. A webMethods Messaging Trigger is trigger that subscribes to and processes documents published/delivered to Universal Messaging, Broker, or locally.
	JMS trigger. A JMS trigger is a trigger that receives messages from a destination (queue or topic) on a JMS provider and then processes those messages.
	Map service.A map service is a service written in the webMethods flow language that allows you to adjust the contents and structure of a pipeline. A map service can be reused in different flow services.
	Provider web service descriptor (WSD). A web service descriptor that contains the definition of a provider IS web service. A provider web service allows an external user to invoke an existing IS service as an "operation" of the web service.
	Consumer web service descriptor (WSD). A web service descriptor that contains the definition of a consumer web service. Consumer web services are external web services that can be invoked from within the local Integration Server.
	Web service connector. A web service connector is a flow service that invokes a web service located on a remote server. Designer automatically generates a web service connector when it creates a web service descriptor for a consumer web service. Designer can also create a web service connector from an existing WSDL.
	Adapter service. An adapter service connects to an adapter's resource and initiates an operation on the resource. Adapter services are created using service templates included with the adapter. For information about creating adapter services, refer to the documentation provided with the adapter.

This icon...	Represents a...
	Adapter notification. An adapter notification enables an adapter to receive event data from the adapter's resource. There are two types of adapter notifications: ■ Polling notifications, which poll the resource for events that occur on the resource. ■ Listener notifications, which work with listeners to detect and process events that occur on the adapter resource. For information about creating an adapter notification, refer to the documentation provided with the adapter.
	Publishable document type for an adapter notification. An adapter notification can have an associated publishable document type that the adapter uses to send the notification data to an Integration Server or a Broker.
	Listener.A listener is an object that connects to an adapter resource and waits for the resource to deliver data when an event occurs on the resource. Listeners work with listener notifications to detect and process event data on the adapter resource. For information about creating a listener, refer to the documentation provided with the adapter.
	Connection. A connection is an object that contains parameters that adapter notifications and listeners use to connect to a resource. For information about creating a connection, refer to the documentation provided with the adapter.
	Flat file dictionary. A flat file dictionary contains record definitions, field definitions, and composite definitions that can be used in multiple flat file schemas.
	Flat file schema. A flat file schema is the blueprint that contains the instructions for parsing or creating the records in a flat file, as well as the constraints to which an inbound flat file document should conform to be considered valid. Using flat file schemas, you can translate documents into and from flat file formats.
	XSLT service. An XSLT service converts XML data into other XML formats or into HTML, using rules defined in an associated XSLT stylesheet.
	Blaze rule service. A rule deployed to Integration Server by Blaze Advisor. Integration Server generates them as rule services and executes them at run time.
	Trading Networks document type. You can drag and drop a Trading Networks (TN) document type into a process model. The "drop" creates a Receive step in the process, with the subscription set to the TN document type name.
	Unknown Node. The webMethods component used to create/develop the element is not installed on the client machine.
	Unknown Service. The webMethods component used to create this service is not installed on the client machine.

This icon...	Represents a...
	WebSocket Server Endpoint. WebSocket server endpoint captures the properties required for callback services for the life cycle of a WebSocket identified by a path which indicates the location of the endpoint within a server.
	WebSocket Client Endpoint. WebSocket client endpoint is designed to capture information related to connection and the respective callback services thereof. It can connect to the server endpoint identified by a URI in a WeSocket application.

UDDI Registry View Icons

The UDDI Registry view contains icons to represent the UDDI Registry, the registered business entities, and the web services that have been published to the UDDI Registry. The following table identifies these icons.

This icon...	Represents a...
	UDDI Registry Node. Designer displays the URL of the UDDI Registry to which you are connected next to the registry icon. Below the UDDI Registry name, Designer displays all of the business entities registered in that UDDI Registry.
	Business Entity. A business entity is a publisher of web services to the UDDI Registry. Below the business entity name, Designer displays the web services published by that entity.
	Web service. A web service is a software application that can be accessed remotely, using XML-based languages to communicate. From a web service or a WSDL, you can create a consumer web service descriptor and connector. Designer can invoke the connector to run the remote web service. From an existing IS service or WSDL, you can create a provider web service descriptor. You can then publish the web service descriptor to a UDDI Registry so that the IS service it describes can be invoked by an external user as an “operation” of the web service.

Flat File Element Icons

A flat file schema or dictionary contains elements used to define the structure and content of the flat file. Flat file schemas or dictionaries can contain elements that are defined inline or contain references to elements defined in a flat file dictionary.

The following table identifies the icon used for each flat file element

Icon	Element	Description
	Schema Definition	The flat file schema. The schema definition is the root element in the flat file schema. The schema definition cannot be moved or deleted from a flat file schema.

Icon	Element	Description
	Record Definition	A collection of fields, composites, and records.
	Record Reference	A reference to a record definition in a flat file dictionary. **Note:** If "Reference not found" appears after the element name, the referenced record no longer exists.
	Composite Definition	A collection of fields. Fields within a composite are referred to as subfields.
	Composite Reference	A reference to a composite definition in a flat file dictionary. **Note:** If "Reference not found" appears after the element name, the referenced composite no longer exists.
	Field Definition	An atomic piece of data.
	Field Reference	A reference to a field definition in a flat file dictionary. **Note:** If "Reference not found" appears after the element name, the referenced field no longer exists.

Flow Step Icons

A flow step is a basic unit of work (expressed in the webMethods flow language) that Integration Server interprets and executes at run time. The webMethods flow language provides flow steps that invoke services and flow steps that let you edit data in the pipeline.

The following table identifies the icon used for each flow step.

Icon	Step	Description
	INVOKE	Executes a specified service.
	MAP	Performs specified editing operations on the pipeline (such as mapping variables in the pipeline, adding variables to the pipeline, and dropping variables from the pipeline).
	BRANCH	Executes a specified flow step based on the value of a specified variable in the pipeline.
	LOOP	Executes a set of flow steps once for each element in a specified array.

Icon	Step	Description
	REPEAT	Re-executes a set of flow steps up to a specified number of times based on the successful or non-successful completion of the set.
	SEQUENCE	Groups a set of flow steps into a series. The SEQUENCE step is implicit in most flow services (that is, the steps in a flow service are treated as a series). However, at times it is necessary to explicitly group a subset of flow steps using SEQUENCE so that they can be treated as a unit.
	EXIT	Controls the execution of a flow step (for example, abort an entire flow service from within a series of deeply nested steps, throw an exception without writing a Java service, or exit a LOOP or REPEAT without throwing an exception).

OData Service Icons

An OData service contains OData elements, properties, and associations used to define the signature of the service.

The following table identifies these icons.

Icon	Description
	Entity Type OData element. A uniquely identifiable OData element that is used to describe the structure of data in the Entity Data Model and can include a key that consists of one or more references to structural properties.
	Complex Type OData element. A structural type OData element consisting of a list of properties but with no key, used to represent entities in the OData service.
	External Entity Type OData element. External entity types are entity types available through an external source provider. For example, if you choose to use webMethods Adapter for JDBC as your external source provider or source type, you can use Database Tables as entity types to create your entities.
	Simple OData property. An OData element property that can contain primitive data (such as a string, an integer, or a Boolean value) that is declared as part of the definition of an OData element.
	Complex OData property. An OData element property that can contain structured data such as a Complex Type.
	Key OData property. Each Entity Type should have at least one **Key** property.

Icon	Description
	Single OData association. An OData association that represents a unidirectional relationship between two entity types.
	Bidirectional OData association. An OData association that represents a bidirectional relationship between two entity types.
	Navigation property. An OData element that defines the relationship between two Entity Types.
	Sync OData property. An OData property that allows you to edit the external entities in an OData service.

REST API Descriptor Icons

A REST API descriptor contains REST resources that identify the operations available in a REST resource. The following table identifies the icons used for REST resources and operations.

Icon	Name	Description
	REST resource	A folder that contains the services that act as REST resources.
	REST V2 resource	A type of REST resource that contains operations defined with a URL template and HTTP methods.
	Parameters	Input parameters for the operation in a REST resource.
	Responses	Responses returned by the operation.
	Group by Tags	Groups the REST resources based on the tags associated with each operation.

Schema Component Icons

A schema is a free-standing element in the Integration Server namespace that is used to validate XML documents. Within Integration Server and Designer, a schema is often referred to as an IS schema to differentiate it from XML schema and flat file schema.

When you select a schema in the Package Navigator view, the Schema Browser lists all of the components of the schema. The following table identifies the icons used for each schema component.

Symbol	Description
	ELEMENTS. Used to group together the global element declarations in an IS schema. This symbol and category name do not correspond to a component in an XML Schema definition or a DTD.
	Element declaration. An element declaration associates an element name with a type definition. This symbol corresponds to the `<element>` declaration in an XML schema definition and the ELEMENT declaration in a DTD.
	Element reference. An element reference is a reference from an element declaration in a content specification to a globally declared element. In an IS schema generated from an XML schema, this symbol corresponds to the `ref="globalElementName"` attribute in an `<element>` declaration. In an IS schema generated from DTD, this symbol appears next to an element that is a child of another element. The parent element has only element content.
	Any element declaration. In XML schema definition, an `<any>` element declaration is a wildcard declaration used as a placeholder for one or more undeclared elements in an instance document. In a DTD, an element declared to be of type ANY can contain any well-formed XML. This symbol corresponds to an element declared to be of type ANY. Because an `<any>` element declaration does not have a name, the Schema Browser uses 'Any' as the name of the element.
	ATTRIBUTES. At the top level of the Schema browser, used to group together the global attribute declarations in an IS schema. Under a complex type definition, used to group together the attribute declarations or attribute references in a complex type definition. This symbol and category name do not correspond to a component in an XML Schema definition or a DTD.
	Attribute declaration. An attribute declaration associates an attribute name with a simple type definition. This symbol corresponds to the XML schema `<attribute>` declaration or the attribute in a DTD ATTLIST declaration.
	Attribute reference. An attribute reference is a reference from a complex type definition to a globally declared attribute. This symbol corresponds to the `ref="globalAttributeName"` attribute in an attribute declaration. DTDs do not have attribute references. Consequently, attribute references do not appear in IS schemas generated from DTDs.

Symbol	Description
	Any attribute declaration. An any attribute declaration is a wildcard declaration used as a placeholder for undeclared attributes in an instance document. This symbol corresponds to the `<anyAttribute>` declaration in an XML schema definition. Because an `<anyAttribute>` declaration does not specify an attribute name, the Schema Browser uses 'Any' as the name of the attribute.
	SIMPLE TYPES. Used to group together the global simple type definitions in an IS schema. This symbol and category name do not correspond to a component in an XML Schema definition or a DTD.
	Simple type definition. A simple type definition specifies the data type for a text-only element or an attribute. Unlike complex type definitions, simple type definitions cannot carry attributes. This symbol corresponds to the `<simpleType>` element in an XML schema definition. If the simple type definition is unnamed (an anonymous type), the Schema Browser displays 'Anonymous' as the name of the simple type definition.
	COMPLEX TYPES. Used to group together the global complex type definitions in an IS schema. This symbol and category name do not correspond to a component in an XML Schema definition or a DTD.
	Complex type definition. A complex type definition defines the structure and content for elements of complex type. (Elements of complex type can contain child elements and carry attributes.) This symbol corresponds to the `<complexType>` element in an XML schema definition. If the complex type definition is unnamed (an anonymous type), the Schema Browser displays 'Anonymous' as the name of the complex type definition.
	Sequence content model. A sequence content model specifies that the child elements in the instance document must appear in the same order in which they are declared in the content model. This symbol corresponds to the `<sequence>` compositor in an XML schema or a sequence list in an element type declaration in a DTD.
	Choice content model. A choice content model specifies that only one of the child elements in the content model can appear in the instance document. This symbol corresponds to the `<choice>` compositor in an XML schema definition or a choice list in a DTD element type declaration.
	All content model. An all content model specifies that child elements can appear once, or not at all, and in any order in the instance document.

Symbol	Description
	This symbol corresponds to the `<all>` compositor in an XML schema definition.
	Mixed content. Elements that contain mixed content allow character data to be interspersed with child elements. This symbol corresponds to the `mixed="true"` attribute in an XML schema complex type definition or a DTD element list in which the first item is `#PCDATA`.
	Empty content. In an XML schema, an element has empty content when its associated complex type definition does not contain any element declarations. An element with empty content may still carry attributes. In a DTD, an element has empty content when it is declared to be of type `EMPTY`.

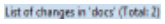
55 Toolbars

- Compare Editor Toolbar 1102
- Document Type Editor Toolbar 1102
- Flat File Schema and Dictionary Editors Toolbars 1103
- Package Navigator View Toolbar 1103
- Pipeline View Toolbar 1103
- REST API Descriptor Toolbar 1105
- Service Editor Toolbar 1105
- Results View Toolbar 1106
- Specification Editor Toolbar 1107
- UDDI Registry View Toolbar 1107
- Variables View Toolbar 1108
- Web Service Descriptor Editor Toolbar 1108

This topic describes the various toolbar buttons available in the Service Development perspective.

Compare Editor Toolbar

The following buttons and message appear on the compare editor toolbar.

Button	Description
	Takes you to the previous different in the change list.
	Takes you to the next difference in the change list.
	Merges changes from left to right.
	Merges changes from right to left.
	Toolbar message which indicates name(s) of the element(s) being compared and the total number of leaf node differences.

Document Type Editor Toolbar

The following buttons appear on the document type editor toolbar.

Button	Description
	Deletes the selected variable. If you select a variable that has children, the children are deleted as well. Equivalent to **Edit > Delete**.
	Moves the selected variable up one position. If the selected variable cannot be moved up from its current position, this button is not available. (Try promoting or demoting it first.)
	Moves the selected variable down one position. If the selected variable cannot be moved down from its current position, this button is not available. (Try promoting or demoting it first.)
	Promotes the selected variable one position to the left. You use this button to move a variable out of a document or document list. If the variable cannot be shifted left from its current position, this button is not available.
	Demotes the selected variable one position to the right. You use this button to make a variable a member of a document or document list. If the variable cannot be shifted right from its current position, this button is not available.
	Displays a list of data types that you can use to create variables for the document type. To create a variable for the document type, select the appropriate data type from this list, and then give the new variable a name.

Flat File Schema and Dictionary Editors Toolbars

The following buttons appear on the flat file schema editor toolbar and on the flat file dictionary editor toolbar.

Button	Description
	Adds a new element.
	Deletes the selected element.
	Moves the selected element up.
	Moves the selected element down.
	Promotes the selected element in a parent/child hierarchy.
	Demotes the selected element in the parent/child hierarchy.
	Creates a document type from the flat file schema. **Note:** This toolbar button appears on the flat file schema editor toolbar only.

Package Navigator View Toolbar

The following buttons appear on the Package Navigator view toolbar.

Button	Description
	Collapses all of the expanded Integration Servers in Package Navigator view.
	Opens the editor for the selected element in Package Navigator view.
	Filters the contents of Package Navigator view by displaying elements of a specified type only.
	Opens Integration Servers preferences where you can add, edit, or remove Integration Server definitions.
	Sorts the contents of Package Navigator view alphabetically by name or alphabetically by element type.
	Runs a service or an element.

Pipeline View Toolbar

The following buttons appear on the Pipeline view toolbar.

Button	Description
	Creates a link between the variables in the pipeline. Links can be created between variables in **Pipeline In** and **Service In**, or between **Service Out** and **Pipeline Out**. In a MAP step, links can be created between variables in **Pipeline In** and **Transformers**, or between **Transformers** and **Pipeline Out**. To link a variable from one stage to another, select the two variables, and then click the link button. This button is not available unless you select two variables that can be linked successfully.
	Creates a **ForEach** mapping between the array variables in the pipeline. **ForEach** mapping can be created only between array variables. To create a **ForEach** mapping, select the two variables, and then click the **ForEach** mapping button. This button is not available unless you select two array variables of the same data type.
	Assigns a value to the selected variable in the **Service In** or **Pipeline Out** stage.
	Drops the selected variable from the pipeline. You may remove a variable from the **Pipeline In** or **Pipeline Out** stage. When you drop a variable, that variable is removed permanently from the pipeline and is not available to subsequent services in the flow.
	Creates a link to or from the value at a specific position in an array variable. This button is available only when at least one of the variables connected by a link is an array variable.
	Adds a new variable to the selected stage in the pipeline. This is a useful way to create a variable that is used by a service in a flow (that is, taken as input or produced as output), but was omitted from the input/output signature.
	Moves the selected variable up one position. If the selected variable cannot be moved up from its current position, this button is not available. (Try promoting or demoting it first.)
	Moves the selected variable down one position. If the selected variable cannot be moved down from its current position, this button is not available. (Try promoting or demoting it first.)
	Promotes the selected variable one position to the left. If the variable cannot be shifted left from its current position, this button is not available.
	Demotes the selected variable one position to the right. If the variable cannot be shifted right from its current position, this button is not available.
	Inserts a service for use as a transformer in a MAP step. This feature is only available when a MAP step is selected in the editor.
	Enables the **Pipeline In**, **Pipeline Out**, **Service In**, **Service Out**, and **Transformers** columns to be scrolled horizontally and vertically independent each other. Independent scrolling is especially useful when mapping a large amount of data in the Pipeline view.

REST API Descriptor Toolbar

The following buttons that are unique to Designer appear on the REST API descriptor editor toolbar.

Button	Description
	Deletes the selected REST resource from the REST API descriptor.
	Adds a REST resource to a REST API descriptor.

Service Editor Toolbar

The following buttons that are unique to Designer appear on the flow service and Java service editor toolbar.

Button	Description
	Deletes the selected flow step. If you select a step that has children, the children are deleted as well. Equivalent to **Edit > Delete**.
	Moves the selected flow step up in the list. If the selected step cannot be moved up from its current position, this button is not available. (Try promoting or demoting it first.)
	Moves the selected flow step down in the list. If the selected step cannot be moved down from its current position, this button is not available. (Try promoting or demoting it first.)
	Promotes a flow step in the parent/child hierarchy. This action moves the step up one level in the hierarchy.
	Demotes a flow step in the parent/child hierarchy. This action makes the selected step a child of the preceding parent step. (This button is only available when you select a step that can become a child.)
	Inserts the previously inserted flow step. Click the ▾ button next to to view the list of flow steps and a list of commonly used services that can be inserted into the flow service as an INVOKE step. You can edit the **Window > Preferences > Software AG > Service Development > Flow Service Editor** preferences to customize this list of services to suit your needs.
	Inserts a MAP step into the flow service. A MAP step performs specified editing operations on the pipeline (for example, adding or dropping variables to or from the pipeline).
	Inserts a BRANCH step into the flow service. A BRANCH step executes a specified step based on the value of a specified variable in the pipeline.

Button	Description
	Inserts a LOOP step into the flow service. A LOOP step executes a set of steps once for each element in a specified array.
	Inserts a REPEAT step into the flow service. A REPEAT step re-executes a set of steps up to a specified number of times based on the successful or non-successful completion of the set.
	Inserts a SEQUENCE step into the flow service. A SEQUENCE step groups a set of steps into a series. The SEQUENCE step is implicit in most cases where a group of flow steps are listed one after another. However, it is often useful to explicitly insert the SEQUENCE step to override default operating parameters. It is also useful for setting up the branches of a BRANCH step.
	Inserts an EXIT step into the flow service. An EXIT step controls the execution of a flow steps; for example, halt an entire flow from within a series of deeply nested steps, throw an exception without writing a Java service, or exit a LOOP or REPEAT without throwing an exception.
	Inserts an INVOKE step into the flow service. Select from the displayed list of services or browse to select a service.

Results View Toolbar

The following buttons appear on the Results view toolbar.

Button	Description
	Reruns launch configurations using same input data.
	Removes the selected result from Results view.
	Removes all results displayed in Results view.
	Goes to the flow step at which an error occurred while running or debugging a flow service.
	Saves the service results pipeline to a file in your local file system.
	Saves the service results pipeline to the *IntegrationServer_directory*instances*instance_name*\pipeline directory on the machine that hosts Integration Server.
	Restores the pipeline contents from a file on your local file system.
	Restores the pipeline contents from the *IntegrationServer_directory*\instances*instance_name*\pipeline directory on the machine that hosts Integration Server.

Button	Description
	Pins a result to Results view so that the result is not removed from Results view.
	Sorts the results in the history pane of Results view alphabetically by element names.

Specification Editor Toolbar

The following buttons that are unique to Designer appear on the specification editor toolbar.

Button	Description
	Deletes the selected variable. If you select a variable that has children, the children are deleted as well. Equivalent to **Edit > Delete**.
	Moves the selected variable up one position. If the selected variable cannot be moved up from its current position, this button is not available. (Try promoting or demoting it first.)
	Moves the selected variable down one position. If the selected variable cannot be moved down from its current position, this button is not available. (Try promoting or demoting it first.)
	Promotes the selected variable one position to the left. You use this button to move a variable out of a document or document list. If the variable cannot be shifted left from its current position, this button is not available.
	Demotes the selected variable one position to the right. You use this button to make a variable a member of a document or document list. If the variable cannot be shifted right from its current position, this button is not available.
	Displays a list of data types that you can use to create variables for the specification. To create a variable for the specification, select the appropriate data type from this list, and then give the new variable a name.

UDDI Registry View Toolbar

The following buttons appear on the UDDI Registry view toolbar.

Button	Description
	Connect to a UDDI Registry while working in Designer.
	Disconnect from a UDDI Registry while working in Designer.
	Refresh the display of web services.

Button	Description
	Copy a web service.
	Publish a copied web service to the selected business entity in the UDDI Registry view.
	Create an expression that filters the contents of the UDDI Registry view based on the value of a web service property.
	Remove the filter from the contents of the UDDI Registry view and display all the published web services.
	Create a web service descriptor (WSD) from the web service selected in the UDDI Registry view.

Variables View Toolbar

The following buttons that are unique to Designer appear on the Variables view toolbar.

Button	Description
	Drop a selected variable from the pipeline passed to the next step in a debugging session.
	Deletes a row in a table for the selected variable.
	Saves the pipeline as an XML document to your local file system.
	Loads a pipeline from a local file. The pipeline you load completely replaces the current debugging pipeline.
	Saves the pipeline as an XML file on in the *IntegrationServer_directory*instances*instance_name*\pipeline directory on the machine that hosts the Integration Server.
	Loads a pipeline from a file in the *IntegrationServer_directory*instances*instance_name*\pipeline directory on the machine that hosts the Integration Server.

Web Service Descriptor Editor Toolbar

The following buttons appear on the web service descriptor (WSD) editor toolbar.

Button	Description
	Adds one or more operations to a web service descriptor.

Button	Description
	Enabled only for provider web service descriptors created from an Integration Server service (in other words, you cannot add an operation to a consumer web service descriptor or a provider web service descriptor created from a WSDL URL or a UDDI Registry). The operation is added to the Operations tab as well as the Binders tab.
	Adds a binder definition (SOAP/Transport/Use-Style) to a web service descriptor. Enabled only for provider web service descriptors created from an Integration Server service (in other words, you cannot add a binder definition to a consumer web service descriptor or a provider web service descriptor created from a WSDL URL or a UDDI Registry). All existing operations are duplicated within the new binder.
	Adds a header or fault element to a Request or Response. If a fault element is selected when you click this button, the default Designer document selector dialog enables you to add a document for the fault. If a header element is selected when you click this button, a special document selector displays only those document types supported by the header handlers listed in the Handler tab.
	Add a header handler to a web service descriptor. Opens a drop-down list of the existing header handlers. The selected handler is added to the top of the list.
	Attach a policy to a web service descriptor.
	Analyze web service descriptor for WS I conformance.
	Regenerate the specified web service connectors. Enabled only for consumer web service descriptors. When no operations are selected, this button refreshes (regenerates) all web service connectors in the web service descriptor. When one or more operations are selected, only the web service connectors for the selected operations are refreshed. This process overwrites existing web service connectors.

56 Keyboard Shortcuts

You can use the following keyboard shortcuts to navigate and perform actions in the Service Development perspective.

Command	Windows and Linux	Mac
Collapse	CTRL+Minus sign (-)	ALT+Left
Debug flow service	CTRL+D	COMMAND+D
Drop	CTRL+SHIFT+O	COMMAND+SHIFT+O
Expand	CTRL+Plus sign (+)	ALT+Right
Locate element	ALT+SHIFT+L	ALT+SHIFT+L
Move Up	CTRL+SHIFT+ Up Arrow	COMMAND+SHIFT+UP
Move Down	CTRL+SHIFT+ Down Arrow	COMMAND+SHIFT+DOWN
Move Left	CTRL+SHIFT+ Left Arrow	COMMAND+SHIFT+LEFT
Move Right	CTRL+SHIFT+ Right Arrow	COMMAND+SHIFT+RIGHT
Open element in editor	CTRL+SHIFT+A	COMMAND+SHIFT+A
Rename	F2	FUNCTION (fn)+F2
Run service	CTRL+R	COMMAND+R
Run in browser	CTRL+B	COMMAND+B
Set Value	CTRL+SHIFT+V	COMMAND+SHIFT+V

57 Conditional Expressions

- Guidelines for Writing Expressions and Filters 1114
- Syntax 1114
- Operators for Use in Conditional Expressions 1117
- Operator Precedence in Conditional Expressions 1123
- Addressing Variables 1124
- Rules for Use of Expression Syntax with the Broker 1126

Integration Server provides syntax and operators that you can use to create expressions for use with the BRANCH step, pipeline mapping, and in trigger conditions.

- In a BRANCH step, you can use an expression to determine the child step that webMethods Integration Server executes. At run time, the first child step whose conditional expression evaluates to "true" is the one that will be executed. For more information about the BRANCH step, see "BRANCH" on page 1070.
- In pipeline mapping, you can place a condition on the link between variables. At run time, webMethods Integration Server only executes the link if the assigned condition evaluates to "true." For more information about applying conditions to links between variables, see "Linking Variables Conditionally" on page 270.
- For webMethods Messaging Triggers, you can further specify the documents that a trigger receives and processes by creating filters for the publishable document types. A filter specifies criteria for the contents of a document.

  Note:
  The conditional expressions syntax is for filters created for documents received from Broker or locally and for the local filter for a document received from Universal Messaging. For information about the syntax for creating a provider filter on Universal Messaging, see the Universal Messaging documentation.
- For JMS triggers, you can create local filters to further limit the messages a JMS trigger processes. A local filter specifies criteria for the contents of the message body. Integration Server applies a local filter to the message after the JMS trigger receives the message from the JMS provider. If the message meets the filter criteria, Integration Server executes the trigger service specified in the routing rule.

Guidelines for Writing Expressions and Filters

When you write expressions and filters, keep the following points in mind:

- Operators, variable names, and strings are case sensitive.
- White space between the tokens of an expression is ignored.
- Some syntax that is valid on the Integration Server is not valid on the Broker. Broker saves the filter with the document type subscription on the Broker only if the syntax is valid. Filters are always saved on the Integration Server. For more information about using trigger filters when Broker is the messaging provider, see "Creating Filters for Use with webMethods Broker " on page 690.

  For a list and an example of syntax that prevents syntax from being saved on the Broker, see "Rules for Use of Expression Syntax with the Broker" on page 1126.

Syntax

When you create an expression, you need to determine which values to include in the expression. Values can be represented as variable names, regular expressions, numbers, and Strings. The

following table identifies the types of values you can use in an expression and the syntax for each value type.

Value Type	Syntax	Description
Regular Expression	*/regularExpression/*	Pattern-matching string. Use the following syntax for pattern matching of variable values: *variableName = /regularExpression/* **Example** `sku = /^WM[0-9]+/` — **Explanation** Evaluates to true if the *sku* variable has a value that starts with "WM" and is followed by one or more digits (WM001, WM95157)
Variable	*variableName* –OR– *%variableName%*	Variable name. For information about how to use this syntax to address children of other variables or elements of array variables, see "Addressing Variables" on page 1124. **Example** `price` — **Explanation** Value of the *price* variable **Example** `%address/postalCode%` — **Explanation** Value of the *postalCode* variable in the *address* document **Example** `%poItems[0]%` — **Explanation** Value of the first element in the *poItems* list.
String	*"string"* –OR– *'string'*	Literal string. Use this value type to compare the value of a variable to a string. **Example** `"Favorite Customer"` — **Explanation** Value is the literal string "Favorite Customer" **Example** `'Favorite Customer'` — **Explanation** Value is the literal string "Favorite Customer" **Note:** Strings not enclosed in quotes (' or ") are interpreted as variable names.
Number	*number*	Number. The following examples indicate the accepted number formats: **Example** — **Explanation**

Value Type	Syntax	Description	
		`-10, 5, 100`	Integers
		`5.0, 6.02`	Floating point number (java.lang.Double)
		`6.345e+4`	Scientific notation
Null	$null	Variable is null or missing. Typically compared with a variable name to determine if the variable is null or missing from the input data.	
		Example	**Explanation**
		`%quantity% = $null`	Evaluates to true if the *quantity* variable is missing from the input data or is null

Comparing Java Objects to Constants

If you want to create a conditional expression that compares a constrained Java Object to a constant value, you must use the following syntax to represent the constant value:

If the object is constrained as type...	Use this syntax...
Boolean	"true" or "false" **Example:** `%myBoolean%=="true"` The string constant in the expression is case insensitive. For example, the expressions `%myBoolean%=="true"` and `%myBoolean%=="tRUe"` are equivalent.
Byte	*"xx"* **Example:** "10" (for 0X0A)
Character	*"a"* **Example:** "C"
Double	*xxxxxx.x* or *xxxxxx* or *-xxxxxx* **Example:** 123456.0, 123456, -123456
Float	*xxxx.x*or *-xxxx.x* **Example:** 1234.1, -1234.1

If the object is constrained as type...	Use this syntax...
Integer	*xxxxx* or *-xxxxx* **Example:** 12345, -12345
Long	*xxxxxx* or *-xxxxxx* **Example:** 123456 or -123456
Short	*xxx* or *-xxx* **Example:** 123 or -123
Date	"*yyyy-MM-dd HH:mm:ss timezone*" **Example:** "2002-06-25 00:00:00 EDT"

Verifying Variable Existence

Sometimes you might want to create an expression that checks only for the existence of a variable in the pipeline or checks to see whether a variable is null. The following table describes the syntax used to check the pipeline for variable existence.

To check if...	Use this syntax...	Description
Variable exists	`variableName`	Evaluates to true if the specified variable exists in the pipeline and has a non-null value.
Variable does not exist	`!variableName`	Evaluates to true if the specified variable does not exist in the pipeline or is null.

Operators for Use in Conditional Expressions

Expressions can include relational and logical operators.

- Relational Operators are used to compare values to each other.
- Logical Operators are used to combine multiple expressions into a single condition.

Relational Operators

You can use relational operators to compare the value of two fields or you can compare the value of a field with a constant. Integration Server provides two types of relational operators: standard and lexical.

- **Standard Relational Operators** can be used in expressions and filters to compare the contents of fields (variables) with other variables or constants.

- **Lexical Relational Operators** can be used to compare the contents of fields (variables) with string values in trigger filters.

Note:
You can also use standard relational operators to compare string values. However, filters that use standard relational operators to compare string values will not be saved with the trigger subscription on Broker. If the subscription filter resides only on Integration Server, Broker automatically places the document in the subscriber's queue. Broker does not evaluate the filter for the document. Broker routes all of documents to the subscriber, creating greater network traffic between Broker and Integration Server and requiring more processing by Integration Server.

Standard Relational Operators

You can use the standard relational operators to compare the contents of a variable with any type of value (numerical, string, Boolean, dates, etc.) or another variable. When comparing strings using the standard operators, Integration Server uses a binary code point comparison algorithm. In this algorithm, Integration Server compares each byte in the first string with each byte in the second string to determine which string is numerically greater. For example, "A" has a value of 65 and "a" has a value of 97, so "a" is greater than "A". Keep the following points in mind when using standard relational operators to compare strings:

- Integration Server considers A to be the lowest letter and Z to be the highest (for example, A < B, A < Z, B > A, Z > A).
- Integration Server considers lowercase letters to be greater than the matching uppercase letter (for example, a > A, A < a, a < B, c > A).
- If you use a standard relational operator to compare numbers in fields of type String, Integration Server treats the contents in the field as numbers. To stop Integration Server from treating the value as a number, you can put quotes (' or ") around the variable name in the expression. For example `%'var1'%=%'var2'%`.

The following table identifies the standard relational operators you can use in expressions and filters.

Operator	Syntax	Description
=	*a* = *b*	Equal to.

This example...	Evaluates to true if...
`customerID = "webMethods"`	The value of the *customerId* variable is "webMethods."

Operator	Syntax	Description
==	*a* == *b*	Equal to.

This example...	Evaluates to true if...
`sku == "WM001"`	The value of the *sku* variable is "WM001."

Operator	Syntax	Description
!=	*a != b*	Not equal to.

This example...	Evaluates to true if...
`quantity != 0`	The value of the *quantity* variable does not equal 0 (zero).

Operator	Syntax	Description
<>	*a <> b*	Not equal to.

This example...	Evaluates to true if...
`state <> 'ME'`	The value of the *state* variable does not equal ME (Maine).

Operator	Syntax	Description
>	*a > b*	Greater than.

This example...	Evaluates to true if...
`price > 100`	The value of the *price* variable is greater than 100.
`%companyID% > "Acme"`	The value of the *companyID* variable is greater than Acme.

Operator	Syntax	Description
>=	*a >= b*	Greater than or equal to.

This example...	Evaluates to true if...
%totalPrice% >= 100	The value of the *totalPrice* variable is greater than or equal to 100.
`companyID >= "Acme"`	The value of the *companyID* variable is greater than or equal to Acme.

Operator	Syntax	Description
<	*a < b*	Less than.

This example...	Evaluates to true if...
`quantity < 5`	The value of the *quantity* variable is less than 5.
`companyID < "Acme"`	The value of the *companyID* variable is less than Acme.

Operator	Syntax	Description
<=	*a <= b*	Less than or equal to.

This example...	Evaluates to true if...
`unitPrice <= 100`	The value of the *unitPrice* variable is less than or equal to 100.

Operator	Syntax	Description
	`companyID <= "Acme"`	The value of the *companyID* variable is less than or equal to Acme.

Lexical Relational Operators

You can use the lexical relational operators to create filters that compare string values. Keep the following points in mind when using the lexical operators:

- When evaluating filters that contain lexical operators, Integration Server uses the locale collating sequence specified on the Broker to compare the values of the strings. The behavior of lexical operators depends on whether a locale is set for the Broker. If no locale is specified, the lexical relational operators behave like the standard relational operators.

  Note:
  To set the filter collation locale, use My webMethods to change the locale on the Broker Server. You might need to restart the Broker Server for the change to take effect. For more information about administering the Broker Server, see *Administering webMethods Broker*.

- If you use a lexical operator to compare strings in an expression (such as in a BRANCH step or in a pipeline link), Integration Server treats the lexical operators as if they were standard relational operators.
- If you use a lexical operator to compare a value that is not a string with another string value, Integration Server treats the non-string value as an empty string (that is, ""). For example, in the expression (`%myInt% L_EQUALS ""`), the `%myInt%`variable is declared to be of type integer. This expression always evaluates to true because `%myInt%` contains an integer value that Integration Server treats as an empty string ("") when it evaluates the expression.
- If you use a lexical operator to compare numbers in fields of type String, Integration Server treats the numbers as strings.
- Filters that use lexical relational operators to compare string values will be saved with the trigger subscription on the Broker. Filters that use standard relational operators to compare string values will not be saved on the Broker.
- When you view filters on My webMethods, a lexical operator appears as its equivalent standard operator. For example, the expression `%myString% L_EQUALS "abc"` appears as `myString=="abc"`.

The following table describes the lexical operators that you can use in filters.

Operator	Description		
L_EQUALS	Lexical equal to.		
	This example...	**Evaluates to true if...**	

Operator	Description	
	`%myString% L_EQUALS "abc"`	The value of the *myString* variable is abc.
L_NOT_EQUALS	Lexical not equal to.	
	This example...	**Evaluates to true if...**
	`%myString% L_NOT_EQUALS "abc"`	The value of the *myString* variable is not abc.
L_LESS_THAN	Lexical less than.	
	This example...	**Evaluates to true if...**
	`%myString% L_LESS_THAN "abc"`	The value of the *myString* variable is less than abc.
L_LESS_OR_EQUAL	Lexical less than or equal to.	
	This example...	**Evaluates to true if...**
	`%myString% L_LESS_OR_EQUAL "abc"`	The value of the *myString* variable is less than or equal to abc.
L_GREATER_THAN	Lexical greater than.	
	This example...	**Evaluates to true if...**
	`%myString% L_GREATER_THAN "abc"`	The value of the *myString* variable is greater than abc.
L_GREATER_OR_EQUAL	Lexical greater than or equal to.	
	This example...	**Evaluates to true if...**
	`%myString% L_GREATER_OR_EQUAL "abc"`	The value of the *myString* variable is greater than or equal to abc.

Logical Operators

You can use the following logical operators in expressions to create conditions consisting of more than one expression:

Operator	Syntax	Description	
!	*! expr*	Negates the next expression.	
		This example...	**Evaluates to true if...**

Operator	Syntax	Description
		`! (%sku% = "WM001")` — The value of the *sku* variable is not equal to WM001.
not	not *expr*	Negates the next expression.

This example...	Evaluates to true if...
`not (color = "blue")`	The *color* variable is not equal to blue.

|	*expr* | *expr*	Logical *OR. True if either of the expressions is true.

This example...	Evaluates to true if...
`%color% = "blue" | %color% = "red"`	The value of the *color* variable is blue or red.

||	*expr* || *expr*	Logical OR. True if either of the expressions is true.

This example...	Evaluates to true if...
`totalPrice > 1000 || customerID = 'Favorite Customer'`	The value of the *totalPrice* variable is greater than 1000 or the value of the *customerID* variable equals Favorite Customer.

or	*expr or expr*	Logical OR. True if either of the expressions is true.

This example...	Evaluates to true if...
`creditCardNum = $null or cardExpireDate = $null or cardExpireDate <= orderDate`	The value of the *creditCardNum* variable is null or missing or if the value of the *cardExpireDate* variable is null or missing or if the value of the *cardExpireDate* variable is less than or equal to the value of the *orderDate*variable.

&	*expr & expr*	Logical AND. Both expressions must evaluate to true for the entire condition to be true.

This example...	Evaluates to true if...
`%customerID% = 'Favorite Customer' & %sku% = 'WM001'`	The value of the *customerID* variable is Favorite Customer and the value of the *sku* variable is WM001.

Operator	Syntax	Description
&&	*expr && expr*	Logical AND. Both expressions must evaluate to true for the entire condition to be true. **This example...** `quantity >= 20 && totalPrice >= 100` **Evaluates to true if...** The value of the *quantity* variable is greater than or equal to 20 and the value of the *totalPrice* variable is greater than or equal to 100.
and	*expr and expr*	Logical AND. Both expressions must evaluate to true for the entire condition to be true. **This example...** `!color and !size` **Evaluates to true if...** The *color* variable does not exist in the input or is null and the *size* variable does not exist in the input or is null.

Operator Precedence in Conditional Expressions

Integration Server evaluates expressions in a condition according to the precedence level of the operators in the expressions. The following table identifies the precedence level of each operator you can use in an expression.

Precedence Level	Operators
1	`()`
2	`not,!`
3	`=,==, !=, <>, >, >=, <, <=`
4	`and, &, &&`
5	`or, |, ||`

Tips

- To override the order in which expressions in a condition are evaluated, enclose the steps you want evaluated first in parentheses. Integration Server evaluates expressions contained in parentheses first.
- When using relational operators to compare strings, Integration Server considers A to be the lowest letter and Z to be the highest (for example, A < B, A < Z, B > A, Z > A). Integration Server

considers lowercase letters to be greater than the matching uppercase letter (for example, a > A, A < a, a < B, c > A).

Addressing Variables

In an expression, you can refer to the values of variables that are children of other variables and refer to the values of elements in an array variable. To address children of variables or an element in an array, you need to use a directory-like notation to describe the position of the value.

Use this notation…	To…
variableName	Address a variable. **Example:** `state` Variable *state*.
variableName/childVariableName	Address the child variable of a variable (such as a field in a document). **Example:** `%buyerInfo/state%` Variable *state* within IS document type *state*.
arrayVariableName[index]	Address an element in an array. **Example:** `orderItems[0]` Value of the first element in the *orderItems* array.
arrayVariableName[rowIndex][columnIndex]	Address an element in a two-dimensional array (String table). **Example:** `dictionary[1][2]` Value of the element located in the third column of the second row in the *dictionary* array.
duplicateVariableName(index)	Address an occurrence of a variable where there are multiple variables with the same name in the document or pipeline. The index is zero-based. **Example:** `address(1)` Value of the second variable named *address*.
%"variableWithSpecialCharacters"%	Address a variable whose name contains special characters. Variables that contain

Use this notation…	To…
	special characters must be enclosed in quotation marks. **Example:** `%"address(work)"%` Value of the variable named address(work). For more information, see "Addressing Variables that Contain Special Characters" on page 1125.

Notes:

- To view the path to a variable in the pipeline, rest the mouse pointer over the variable name. Designer displays the variable path in a tool tip.
- To copy the path to a variable in a pipeline, select the variable, right-click, and select **Copy**.
- You can enclose variable names in %, for example `%buyerInfo/state%`. If the variable name includes special characters, you *must* enclose the path to the variable in % (percent) symbols and enclose the variable name in " " (quotation marks). For more information about using variables as values in expressions, see "Syntax" on page 1114.

Addressing Variables that Contain Special Characters

If a variable name contains any special characters, you need to use the following notation to address the variable:

- Enclose the path to the variable and the variable name in % (percent symbols).
- Enclose the variable name that contains special characters in " " (quotation marks).

Following are some examples of how to address variables that contain special characters.

Type...	To...
`%"Date/Time"%`	Address a variable named *Date/Time*.
`%purchaseOrder/"Date/Time"%`	Address a variable named *Date/Time* in the document variable *purchaseOrder*.If you did not enclose *Date/Time* in quotation marks, and instead had *%purchaseOrder/Date/Time%* or *%"purchaseOrder/Date/Time"%*, the expression would address a variable named *Time* in a document named *Date* that was contained in a document named *purchaseOrder*.
`%"address(work)"/"phone(cell)"%`	Address a variable named *phone(cell)* in the document variable *address(work)*.

Type...	To...
`%"Date\\Time"%`	Address a variable named *Date\Time*.

Typing Special Characters in Expressions

You enter most of the special characters in an expression just as you would enter them when creating the variable name. However, for four of the special characters (the backslash, slash mark, percent symbol, and quotation marks), you need to use a combination of keys. The following table identifies the special characters for variable names and any key sequences that you need to use to enter a variable name with that character in an expression.

Character	Character Name	Special Sequence
\	backslash	\\
[	opening bracket	
]	closing bracket	
(	opening parenthesis	
)	closing parenthesis	
%	percent	\%
"	quotation marks	\"
/	slash mark (forward)	/\

Note:
When you use variable names with special characters in expressions or filters, you must enclose the variable name in " " (quotation marks).

Rules for Use of Expression Syntax with the Broker

When you create filters for webMethods Messaging Triggers that receive documents from Broker, keep in mind that some syntax that is valid on Integration Server is not valid on Broker. When you save a webMethods Messaging Trigger, Integration Server and Broker evaluate the filter to make sure that it uses proper syntax. If the syntax is valid on Broker, Broker saves the subscription and the filter. If the syntax is invalid on Broker, Integration Server automatically removes the filter before the Broker saves the subscription. The filter will only be saved on Integration Server.

Broker saves as much of a filter as possible with the subscription. For example, suppose that a filter consists of more than one expression, and only one of the expressions contains syntax Broker considers invalid. Broker saves the expressions it considers valid but does not save the expression containing invalid syntax. (Integration Server saves all the expressions.)

Keep the following points in mind when writing filters for webMethods Messaging Triggers:

- Expressions that specify field names that contain syntax, characters, symbols, or words the Broker considers restricted or reserved will not be saved on the Broker.
- All expressions must contain a relational (comparison) operator.
- Use lexical relational operators (such as L_EQUALS, L_LESS_THAN) to compare fields of type String.
- Use standard relational operators (such as =, ==, !=, <, >, <= and >=) to compare fields that are not of type String.
- Use the =, ==, <>, or != operators to compare a value with an Object constrained as a Boolean.
- You can use My webMethods to view the filters (expressions) saved with a subscription. If the expression does not appear with the subscription on the Broker, then the expression contains invalid syntax.

The following table identifies syntax that the Broker considers invalid. Expressions with this syntax will be saved on Integration Server but not on the Broker.

Broker considers expressions invalid when they contain....	**Examples**
Field names with syntax, characters, symbols, or words the Broker considers restricted or reserved	`eventtype L_EQUALS “addEmployee”tax% < 5` Although Broker considers a field name that contains the % symbol to be invalid, you can use the % symbol to enclose field names in the expression.
No comparison operators	`"fieldName" "!fieldName"`
A standard relational operator to compare fields of type String	`%myString% < "yourString"`
A lexical relational operator to compare fields that are not of type String	`%price% L_LESS_THAN 50`
A field of type String compared with a numeric value	`"stringName" > 12`
Operators other than =, ==, !=, or <> to compare an Object constrained as a Boolean with a value	`myBoolean <= "true"`
An Object constrained as a Boolean compared with a field of type String	`myBoolean = "stringFieldName"` **Note:** Expressions that check an Object constrained as a Boolean for a true or false value should include “true” or “false” as part of the filter. The string constant in the expression (“true” or “false”) is case insensitive.

Broker considers expressions invalid when they contain....	Examples
A $null token	`%fieldName% = $null`
Regular expressions that contain back references	`fieldName = /^(a|b)\1$/`
Regular expressions that use quantifiers other than +, ?, and *	`/a{1}/ /a{1,5}/`
Regular expressions that use extended metacharacters	`fieldName = /\w/`

58 Regular Expressions

- Using a Regular Expression in a Mask 1130
- Regular Expression Operators 1130

A *regular expression* is a pattern-matching technique used extensively in UNIX environments. You use regular expressions in Designer to specify pattern-matching strings for some of its functions. For example, you can use a regular expression to specify an index, a property, or a mask in a webMethods Query Language (WQL) statement. You can also use a regular expression to specify the switch value for a BRANCH step.

Note: Integration Server and Designer use PERL regular expressions by default.

To specify a regular expression, you must enclose the expression between / symbols. When the server encounters this symbol, it knows to interpret the characters between these symbols as a pattern-matching string (that is, a regular expression).

A simple pattern-matching string such as /string/ matches any element that contains string. So, for example, the regular expression /webMethods/ would match all of the following strings:

```
"webMethods"
"You use webMethods Integration Server to execute services"
"Exchanging data with XML is easy using webMethods"
"webMethods Integration Server"
```

Important:
Characters in regular expressions are case sensitive.

Using a Regular Expression in a Mask

When you use a regular expression as a mask, you use parentheses to specify which characters you want to collect. For example, the object reference:

```
doc.p[].text[/(.{30}).*/]
```

retains the first 30 characters in each matching element and discards the rest.

Regular Expression Operators

Following are the operators supported in the webMethods implementation of regular expressions.

Use this symbol…	To…
.	Match any single character except a new line. **Example** `doc.p[/web.ethods/].text` This example would return any paragraph containing the string ‘web’ followed by any single character and the string ‘ethods’. It would match both ‘webMethods’ and ‘webmethods’.
^	Match the beginning of the string or line. **Example** `doc.p[/^webMethods/].text`

Use this symbol…	To…
	This example would return any paragraph containing the string 'webMethods' at the beginning of the element or at the beginning of any line within that element.
$	Match the end of the string or line. **Example** `doc.p[/webMethods$/].text` This example would return any paragraph containing the string 'webMethods' at the end of the paragraph element or at the end of any line within that element.
*	Match the preceding item zero or more times. **Example** `doc.p[/part *555-A/].text` This example would return any paragraph containing the string 'part' followed by zero or more spaces and then the characters '555-A'.
+	Match the preceding item 1 or more times. **Example** `doc.p[/part +555-A/].text` This example would return any paragraph containing the string 'part' followed by one or more spaces and then the characters '555-A'.
?	Match the preceding item 0 or 1 times. **Example** `doc.p[/part ?555-A/].text` This example would return any paragraph containing the string 'part' followed by zero or one space and then the characters '555-A'.
( )	When used in an index, these characters group an item within the regular expression. **Example** `doc.p[/part(,0)+May/].text` This example would return any paragraph containing the string 'part' followed by one or more occurrences of the characters ',0' and then the characters 'May'. When used in a mask, they specify characters that you want to retain. **Example** `doc.p[].text[(^.{25}).*]` This example would keep the first 25 characters within each paragraph and discard the rest.
{*n* }	Match the preceding item exactly *n* times. **Example** `doc.p[/^.{24}webmethods/].text`

Use this symbol…	To…
	This example would return any paragraph in which the word ‘webmethods’ started in the 25th character position of the paragraph.
{*n* ,}	Match the preceding item *n* or more times.
	Example `doc.p[/^.{10,}webmethods/].text`
	This example would return any paragraph in which the word ‘webmethods’ appeared anywhere after the 10th character position of the paragraph. That is, this example would return a paragraph in which the word ‘webmethods’ started in the 11th or later character position of the paragraph.
{0,*m*}	Match the preceding item none or at most *m* times.
	Example `doc.p[/^.{1,4}webmethods/].text`
	This example would return any paragraph in which the word ‘webmethods’ started in character position 2 through 5 of the paragraph.
|	Match the expression that precedes or follows this character.
	Example `doc.p[/webmethods|webMethods/].text`
	This example would return any paragraph that contained either ‘webmethods’ or ‘webMethods’.
\b	Match a word boundary.
	Example `doc.p[/\bport\b/].text`
	This example would return any paragraph that contained the word ‘port’, but not paragraphs that contained these characters as part of a larger word, such as ‘import’, ‘support’, ‘ports’ or ‘ported’.
\B	Match a boundary that is not a word boundary.
	Example `doc.p[/\B555-A/].text`
	This example would return any paragraph that contained the characters ‘555-A’ as part of a larger word such as AZ555-A, or Dept555-A, but not ‘555-A’ alone.
\A	Match only at the beginning of a string.
	Example `doc.p[/\AwebMethods/].text`
	This example would return any paragraph containing the string ‘webMethods’ at the beginning of the element or at the beginning of any line within that element.
\Z	Match only at the end of a string (or before a new line at the end).

Use this symbol…	To…
	Example `doc.p[/webMethods\Z/].text` This example would return any paragraph containing the string 'webMethods' at the end of the paragraph element or at the end of any line within that element.
\n	Match a new line. **Example** `doc.p[/webMethods\n/].text` This example would return any paragraph containing the string 'webMethods' followed by the new line character.
\r	Match a carriage return. **Example** `doc.p[/webMethods\r/].text` This example would return any paragraph containing the string 'webMethods' followed by a carriage return.
\t	Match a tab character. **Example** `doc.p[/\twebMethods/].text` This example would return any paragraph containing the string 'webMethods' preceded by a tab character.
\f	Match a form feed character. **Example** `doc.p[/webMethods\f/].text` This example would return any paragraph containing the string 'webMethods' followed by a form feed character.
\d	Match any digit. Same as [0-9]. **Example** `doc.p[/part \d555-A/].text` This example would return any paragraph containing a part number that starts with any digit 0 through 9, and is followed by the characters 555-A. Therefore, it would match 'part 1555-A' but not 'part A555-A' or 'part #555-A'.
\D	Match any non-digit. Same as [^0-9]. **Example** `doc.p[/part \D555-A/].text` This example would return any paragraph containing a part number that starts with any character other than 0 through 9, and is followed by the characters 555-A. Therefore, it would match 'part A555-A' and 'part #555-A', but not 'part 1555-A'.
\w	Match any word character. Same as [0-9a-z_A-Z].

Use this symbol…	To…
	Example `doc.p[/part \w4555-A/].text` This example would return any paragraph containing a part number that starts with a letter or digit and is followed by the characters 555-A. Therefore, it would match 'part A555-A' and 'part 1555-A', but not 'part #555-A'.
\W	Match any nonword character. Same as [^0-9a-z_A-Z]. **Example** `doc.p[/part \W4555-A/].text` This example would return any paragraph containing a part number that starts with a character other than a letter or digit, and is followed by the characters 555-A. Therefore, it would match 'part #555-A' and 'part -555-A', but not 'part 1555-A' or 'part A555-A'.
\s	Match any white-space character. Same as [\t\n\r\f]. **Example** `doc.p[/\swebMethods/].text` This example would return any paragraph containing the string 'webMethods' if it is preceded by a tab character, a new line character, a carriage return, or a form-feed character.
\S	Match any nonwhite-space character. Same as [^\t\n\r\f]. **Example** `doc.p[/\SwebMethods/].text` This example would return any paragraph containing the string 'webMethods', if that string is not preceded by a tab character, a new line character, a carriage return, or a form-feed character.
\0	Match a null string. **Example** `doc.p[/[^\0]/].text` This example would return any paragraph that is not empty (null).
\x*nn*	Match any character with the hexadecimal value *nn*. **Example** `doc.p[/\x1FwebMethods/].text` This example would return any paragraph containing the ASCII unit-separator character (1F) followed by the characters 'webMethods'. This example would return any paragraph containing the ASCII unit-separator character (1F) followed by the characters 'webMethods'.
[ ]	Match any character within the brackets. **Example** `doc.p[/part [023]555-A/].text`

Use this symbol…	To…

This example would return any paragraph containing a part number that starts with the numbers 0, 2, or 3 and is followed by the characters 555-A. Therefore, it would match ‘part 0555-A’ and ‘part 2555-A’, but not ‘part 4555-A’.

The following characters have special meaning when used within brackets:

number that starts with the numbers 0, 2, or 3 and is followed by the characters 555-A. Therefore, it would match ‘part 0555-A’ and ‘part 2555-A’, but not ‘part 4555-A’.

The following characters have special meaning when used within brackets:

Use this char…	To…
^	Exclude characters from the pattern. **Example** `doc.p[/part [^023]555-A/].text` This example would return any paragraph containing a part number that does not start with the numbers 0, 2, or 3, but is followed by the characters 555-A. Therefore, it would match ‘part 4555-A’ and ‘part A555-A’, but not ‘part 0555-A’.
-	Specify a range of allowed characters. **Example** `doc.p[/part [A-M]555-A/].text` This example would return any paragraph containing a part number that starts with any letter A through M and is followed by the characters 555-A. Therefore, it would match ‘part A555-A’ and ‘part J555-A’, but not ‘part N555-A’.

59 Validation Content Constraints

- Content Types 1138
- Constraining Facets 1148

You can apply content constraints to variables in the IS document types, specifications, or service signatures that you want to use as blueprints in data validation. Content constraints describe the data a variable can contain. At validation time, if the variable value does not conform to the content constraints applied to the variable, the validation engine considers the value to be invalid. For more information about validation, see "Performing Data Validation" on page 293.

When applying content constraints to variables, you can do the following:

- **Select a content type.** A content type specifies the type of data for the variable value, such as string, integer, boolean, or date. A content type corresponds to a simple type definition in a schema.
- **Set constraining facets.** Constraining facets restrict the content type, which in turn, restrict the value of the variable to which the content type is applied. Each content type has a set of constraining facets. For example, you can set a length restriction for a string content type, or a maximum value restriction for an integer content type.

For example, for a String variable named *itemQuantity*, you might specify a content type that requires the variable value to be an integer. You could then set constraining facets that limit the content of *itemQuantity* to a value between 1 and 100.

The content types and constraining facets described in this appendix correspond to the built-in data types and constraining facets in XML Schema. The World Wide Web Consortium (W3C) defines the built-in data types and constraining facets in the specification *XML Schema Part 2: Datatypes* (http://www.w3c.org/TR/xmlschema-2).

Content Types

The following table identifies the content types you can apply to String, String list, or String table variables. Each of these content types corresponds to a built-in simple type defined in the specification *XML Schema Part 2: Datatypes*.

Note:
For details about constraints for Objects and Object lists, see "Data Types" on page 1083.

Content Types	Description
anyURI	A Uniform Resource Identifier Reference. The value of anyURI may be absolute or relative. **Constraining Facets** enumeration, length, maxLength, minLength, pattern **Note:** The anyURI type indicates that the variable value plays the role of a URI and is defined like a URI. webMethods Integration Server does not validate URI references because it is impractical for applications to check the validity of a URI reference.
base64Binary	Base64-encoded binary data.

Content Types	Description
	Constraining Facets enumeration, length, maxLength, minLength, pattern
boolean	True or false. **Constraining Facets** pattern **Example** `true, 1, false, 0`
byte	A whole number whose value is greater than or equal to –128 but less than or equal to 127. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `-128, -26, 0, 15, 125`
date	A calendar date from the Gregorian calendar. Values need to match the following pattern: CCYY-MM-DD Where CC represents the century, YY the year, MM the month, DD the day. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `1997-08-09` (August 9, 1997)
dateTime	A specific instant of time (a date and time of day). Values need to match the following pattern: CCYY-MM-DDThh:mm:ss.sss Where CC represents the century, YY the year, MM the month, DD the day, T the date/time separator, hh the hour, mm the minutes, and ss the seconds. The pattern can include a Z at the end to indicate

Content Types	Description
	Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `2000-06-29T17:30:00-05:00` represents 5:30 pm Eastern Standard time on June 29, 2000. (Eastern Standard Time is 5 hours behind Coordinated Universal Time.)
decimal	A number with an optional decimal point. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `8.01, 290, -47.24`
double	Double-precision 64-bit floating point type. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `6.02E23, 3.14, -26, 1.25e-2`
duration	A length of time. Values need to match the following pattern: P*n*Y*n*M*n*DT*n*H*n*M*n*S Where *n*Y represents the number of years, *n*M the number of months, *n*D the number of days, T separates the date and time, *n*H the number of hours, *n*M the number of minutes and *nS* the number of seconds. Precede the duration with a minus (-) sign to indicate a negative duration. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example**

Content Types	Description
	`P2Y10M20DT5H50M` represents a duration of 2 years, 10 months, 20 days, 5 hours, and 50 minutes
ENTITIES	Sequence of whitespace-separated ENTITY values declared in the DTD. Represents the ENTITIES attribute type from the XML 1.0 Recommendation. **Constraining Facets** enumeration, length, maxLength, minLength
ENTITY	Name associated with an unparsed entity of the DTD. Represents the ENTITY attribute type from the XML 1.0 Recommendation. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace
float	A number with a fractional part. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `8.01, 25, 6.02E23, -5.5`
gDay	A specific day that recurs every month. Values must match the following pattern: ---DD Where DD represents the day. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `---24` indicates the 24th of each month
gMonth	A Gregorian month that occurs every year. Values must match the following pattern: --MM

Content Types	Description
	Where MM represents the month. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `--11` represents November
gMonthDay	A specific day and month that recurs every year in the Gregorian calendar. Values must match the following pattern: --MM-DD Where MM represents the month and DD represents the day. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `--09-24` represents September 24th
gYear	A specific year in the Gregorian calendar. Values must match the following pattern: CCYY Where CC represents the century, and YY the year. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `2001` indicates 2001

Content Types	Description
gYearMonth	A specific month and year in the Gregorian calendar. Values must match the following pattern: CCYY-MM Where CC represents the century, YY the year, and MM the month. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern **Example** `2001-04` indicates April 2001
hexBinary	Hex-encoded binary data. **Constraining Facets** enumeration, length, maxLength, minLength, pattern
ID	A name that uniquely identifies an individual element in an instance document. The value for ID needs to be a valid XML name. The ID datatype represents the ID attribute type from the XML 1.0 Recommendation. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace
IDREF	A reference to an element with a unique ID. The value of IDREF is the same as the ID value. The IDREF datatype represents the IDREF attribute type from the XML 1.0 Recommendation. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace
IDREFS	Sequence of white space separated IDREFs used in an XML document. The IDREFS datatype represents the IDREFS attribute type from the XML 1.0 Recommendation. **Constraining Facets** enumeration, length, maxLength, minLength
int	A whole number with a value greater than or equal to -2147483647 but less than or equal to 2147483647.

Content Types	Description
	Constraining Facets enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `-21474836, -55500, 0, 33123, 4271974`
integer	A positive or negative whole number. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `-2500, -5, 0, 15, 365`
language	Language identifiers used to indicate the language in which the content is written. Natural language identifiers are defined in IETF RFC 1766. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace
long	A whole number with a value greater than or equal to -9223372036854775808 but less than or equal to 9223372036854775807. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `-55600, -23, 0, 256, 3211569432`
Name	XML names that match the Name production of XML 1.0 (Second Edition). **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace
NCName	Non-colonized XML names. Set of all strings that match the NCName production of Namespaces in XML. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace

Content Types	Description
negativeInteger	An integer with a value less than or equal to –1. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `-255556, -354, -3, -1`
NMTOKEN	Any mixture of name characters. Represents the NMTOKEN attribute type from the XML 1.0 Recommendation. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace
NMTOKENS	Sequences of NMTOKENS. Represents the NMTOKENS attribute type from the XML 1.0 Recommendation. **Constraining Facets** enumeration, length, maxLength, minLength
nonNegativeInteger	An integer with a value greater than or equal to 0. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `0, 15, 32123`
nonPositiveInteger	An integer with a value less than or equal to 0. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace **Example** `-256453, -357, -1, 0`
normalizedString	Represents white space normalized strings. Set of strings (sequence of UCS characters) that do not contain the carriage return (#xD), line feed (#xA), or tab (#x9) characters. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace

Content Types	Description
	Example `MAB-0907`
positiveInteger	An integer with a value greater than or equal to 1. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `1, 1500, 23000`
short	A whole number with a value greater than or equal to -32768 but less than or equal to 32767. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `-32000, -543, 0, 456, 3265`
string	Character strings in XML. A sequence of UCS characters (ISO 10646 and Unicode). By default, all white space is preserved for variables with a string content constraint. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace **Example** `MAB-0907`
time	An instant of time that occurs every day. Values must match the following pattern: hh:mm:ss.sss Where hh indicates the hour, mm the minutes, and ss the seconds. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time. **Constraining Facets** enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern

Content Types	Description
	Example `18:10:00-05:00` (6:10 pm, Eastern Standard Time) Eastern Standard Time is 5 hours behind Coordinated Universal Time.
token	Represents tokenized strings. Set of strings that do not contain the carriage return (#xD), line feed (#xA), or tab (#x9) characters, leading or trailing spaces (#x20), or sequences of two or more spaces. **Constraining Facets** enumeration, length, maxLength, minLength, pattern, whiteSpace
unsignedByte	A whole number greater than or equal to 0, but less than or equal to 255. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `0, 112, 200`
unsignedInt	A whole number greater than or equal to 0, but less than or equal to 4294967295. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `0, 22335, 123223333`
unsignedLong	A whole number greater than or equal to 0, but less than or equal to 18446744073709551615. **Constraining Facets** enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits **Example** `0, 2001, 3363124`
unsignedShort	A whole number greater then or equal to 0, but less than or equal to 65535. **Constraining Facets**

Content Types	Description
	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits
	Example `0, 1000, 65000`

Constraining Facets

When you apply a content type to a variable, you can also set constraining facets for the content type. Constraining facets are properties that further define the content type. For example, you can set a minimum value or precision value for a decimal content type. Each content type has a set of constraining facets. The constraining facets described in the following table correspond to constraining facets defined in the specification *XML Schema Part 2: Datatypes*.

Constraining Facet	Description	Usage Notes
enumeration	The possible values for the variable at run time.	If you also entered possible values using the **Pick list choices** property in the **General** category of the Properties view, those values will be displayed at run time. However, the **enumeration** values will be used for validation.
fractionDigits	The maximum number of digits to the right of the decimal point. For example, the **fractionDigits** of the value `999.99` is `2`.	**fractionDigits** must be less than or equal to **totalDigits**.
length	The precise units of length required for the variable value.	If you specify **length**, you cannot specify either **minLength** or **maxLength**.
maxExclusive	The upper bound of a range of possible values. The range excludes the value you specify. The variable can have a value less than but not equal to **maxExclusive**.	**maxExclusive** must be greater than or equal to **minExclusive**. You cannot specify **maxInclusive** and **maxExclusive** for the same content type.
maxInclusive	The upper bound of a range of possible values. The range includes the value you specify. The variable	**maxInclusive** must be greater than or equal to **minInclusive**.

Constraining Facet	Description	Usage Notes
	can have a value less than or equal to **maxInclusive**.	You cannot specify **maxInclusive** and **maxExclusive** for the same content type.
maxLength	The maximum units of length permitted for the variable value.	**maxLength** must be greater than or equal to **minLength**.
minExclusive	The lower bound of a range of possible values. The range does not include the value you specify. The variable can have a value greater than but not equal to **minExclusive**.	**minExclusive** must be less than or equal to **maxExclusive**. You cannot specify **minInclusive** and **minExclusive** for the same content type.
minInclusive	The lower bound of a range of possible values. The range includes the value you specify. The variable can have a value greater than or equal to **minInclusive**.	**minInclusive** must be less than or equal to **maxInclusive**. You cannot specify **minInclusive** and **minExclusive** for the same content type.
minLength	The minimum units of length permitted for the variable value.	**minLength** must be less than or equal to **maxLength**.
pattern	A pattern (regular expression) that the value of the variable must match. For example, you can use a regular expression to specify that a variable that is a string content constraint match a Social Security number format.	
totalDigits	The maximum number of decimal digits allowed in a value. For example, the **totalDigits** of the value `999.99` is `5`.	**totalDigits** must be greater than or equal to **fractionDigits**.
whiteSpace	The white space normalization performed on the variable value. The value of **whiteSpace** can be one of the following: `preserve`: No white space normalization is performed.	

Constraining Facet	Description	Usage Notes
	`replace`: Carriage returns (#xD), line feeds (#xA), and tabs (#x9) are replaced with a single space (#x20). `collapse`: After the white space normalization specified by replace is performed, sequences of spaces (#x20) and leading and trailing spaces (#x20) are removed.	

Note:
Previous versions of XML Schema contained the constraining facets **duration**, **encoding**, **period**, **precision**, and **scale**. However, these constraining facets are not included in the recommendation of *XML Schema Part 2: Datatypes*. The constraining facets **duration**, **encoding**, and **period** were removed. **precision** was renamed **totalDigits**. **scale** was renamed **fractionDigits**. If you view a content type from an IS schema created from an XML Schema Definition that used pre-Recommendation version of XML Schema (before May 2001) the Content Type dialog box will display the constraining facets that were available in the pre-Recommendation version of XML Schema.

Note:
The word "fixed" appears next to the name of a constraining facet whose value is fixed and cannot be changed. When a facet has a fixed value, the facet is called a fixed facet.

60 webMethods Query Language

- Overview 1152
- Object References 1152
- Sibling Operators 1153
- Object Properties 1155
- Property Masking 1155

The webMethods Query Language (WQL) to map data from web documents. This topic describes the WQL and the references, operators, and properties available for use while parsing the contents of web documents.

Overview

The webMethods Query Language (WQL) provides the primary mechanism for mapping data from web documents. When a web document is read by webMethods, the XML or HTML markup within the document is used to parse the contents of the document into the object model.

XML and HTML markup both consist of tag elements enclosed in angle brackets: <>. In the process of parsing, tag elements are transformed into arrays of objects; the attributes of tag elements become object properties. XML and HTML markup both implement containing elements and empty elements. Containing elements have open and close tags. Empty elements are single tags.

When a web document is parsed, the text contained within containing elements becomes the text property of the corresponding XML node.

Data parsed from web documents is accessed with WQL queries, which consist of one or more indexed element arrays and an object property.

Note:
When you use pub.xml:queryXMLNode to query an enhanced XML node (a node produced by the enhanced XML parser), you must use XQL as the query language. WQL cannot be used to query an enhanced XML node.

Object References

For the following object references, x and y represent numerical indexes.

doc.*element*[x].*property*
An absolute reference uses a numerical index into an element array.

doc.*element*[x].element[x].*property*
Nested element arrays scope the object reference to children elements.

doc.*element*[x].*line*[x].*property*
An array of lines is fabricated when the text property of a node contains line breaks.

doc.*element*[x].^.*property*
The parent of an element is referenced with ^.

doc.*element*[x].?[x].*property*
A ? matches any type of element array.

doc.*element*[].*property*
Empty brackets signify that all members of an element array are to be returned.

doc.*element*|*element*[].*property*
The | is used to signify a logical 'OR'. The contents of two or more element arrays can be returned.

doc.*element*[x-y].*property*

Returns a range of elements from an array.

doc.*element*[x-end].*property*
end is a reserved word that returns the final element of an array.

doc.*element*[x,y,z].*property*
Returns items `x`, `y`, and `z` where x, y, and z represent numerical indexes into an element array.

doc.*element*[x+y].*property*
Returns element `x` and every `y` element thereafter.

doc.*element*['match'].*property*
Returns an array of elements whose text property matches the `match` string, which can contain the following wildcard characters:

Use this character...	To...
`*`	Match any sequence of zero or more characters.
`?`	Match any single character.
`%`	Matches a single word, where word is defined to be any sequence of non-whitespace characters.
`\`	Escape any of the above wildcard characters.

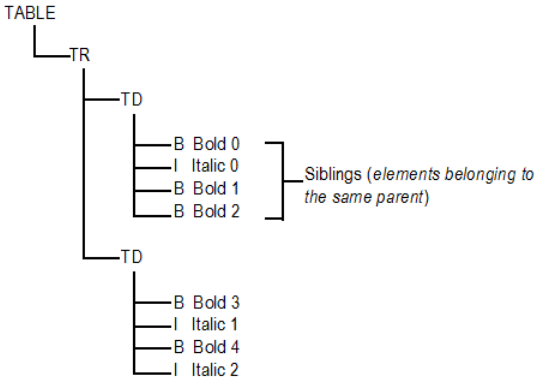
Match strings are compared with the `.text` property of the indexed object. The `.text` property contains the text of all child objects.

doc.*element*[/RegularExpression/].*property*
Returns an array of elements whose text property matches the specified regular expression. For information about how to construct a regular expression, see "Regular Expressions" on page 1129.

doc.*element*(*property*='match').*property*
Matches the value of a specific element property.

Sibling Operators

WQL provides the following set of operators to refer to siblings of a specified element. The examples shown in these descriptions refer to the following HTML structure:



The sibling operators are constrained by the current parent. If an operator exceeds the boundaries of the current parent, a null value is produced for that reference.

doc.*element*[x].@n.*property*
References the *nth* sibling after element[x], regardless of type. Compare with doc.*element*[x].+n.*property*, below.

Example	Result
`doc.td[0].b[0].@1.text`	`Italic 0`
`doc.td[0].i[0].@1.text`	`Bold 1`
`doc.td[0].b[0].@4.text`	`Null`

doc.*element*[x].@-n.*property*
References the *nth* sibling prior to element[x], regardless of type. Compare with doc.*element*[x].-n.*property*, below.

Example	Result
`doc.td[1].b[end].@-2.text`	`Bold 3`
`doc.td[1].i[end].@-1.text`	`Bold 4`
`doc.td[1].b[end].@-3.text`	`Null`

doc.*element*[x].+n.*property*
References the *nth* sibling after element[x] that is of the same type as element[x]. Compare with doc.*element*[x].@n.*property*, above.

Example	Result
`doc.td[0].b[0].+1.text`	`Bold 1`
`doc.td[0].i[0].+1.text`	`Null`
`doc.td[0].b[0].+3.text`	`Null`

doc.*element*[x].-n.*property*
References the *nth* sibling before element[x] that is of the same type as element[x]. Compare with doc.*element*[x].@-n.*property*, above.

Example	Result
`doc.td[1].b[end].-2.text`	`Null`
`doc.td[1].i[end].-1.text`	`Italic 1`
`doc.td[1].b[end].-3.text`	`Null`

Object Properties

In addition to the properties derived from the attributes of a parsed XML or HTML tag element, the following properties are available for all objects:

text/.txt
Returns the text of an object.

.value/.val
Returns the value of an object. (Equivalent to the text of the object if the element has no VALUE attribute.)

.source/.src
Returns the XML or HTML source of an object.

.csource/.csrc
Returns the XML or HTML of the source generated from the parse tree of the document.

.index/.idx
Returns the numerical index of an object.

.reference/.ref
Returns a complete object reference.

Property Masking

Property masking allows for the stripping away of unwanted text from the value of an object property.

doc.*element*[x].*property*[x-y]
Returns a range of characters from position *x* to *y*.

doc.*element*[x].*property*['mask']
Uses wildcard matching and token collecting to extract desired data from the value of an object property.

doc.*element*[x].*property*[/RegularExpression/]
Uses a regular expression to extract desired data from the value of an object property. For information about how to construct a regular expression, see "Regular Expressions" on page 1129.