

Run-Time Governance with CentraSite

Version 9.9

October 2015

This document applies to ContraSite Version 9.9 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 © 2005-2015 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 <http://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 <http://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 <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Table of Contents

About this Guide	15
Document Conventions.....	15
Online Information.....	16
Overview of Run-Time Governance	17
Components of the Run-Time Governance Environment.....	18
Run-Time Gateways.....	19
Virtualized Services in CentraSite Control.....	20
Virtualizing APIs Using the CentraSite Business UI.....	20
Run-Time Governance Reference Information.....	21
Consumer Applications	23
Introduction to Consumer Applications.....	24
Roles and Permissions Needed to Create and Manage Consumer Applications.....	24
Creating and Managing Consumer Applications.....	25
Overview.....	25
Adding and Managing Consumer Applications Using CentraSite Control UI.....	25
Adding a Consumer Application.....	25
Modifying Consumer Application Details.....	26
Configuring Consumer Application Profiles.....	26
The Identification Profile.....	26
The Permissions Profile.....	29
The Versions Profile.....	29
The Subscriptions Profile.....	29
The Audit Log Profile.....	29
The Object-Specific Properties Profile.....	29
Deleting Consumer Applications.....	29
Adding and Managing Consumer Applications Using CentraSite Business UI.....	30
Creating a Consumer Application.....	30
Modifying Consumer Application Details.....	31
Configuring Consumer Application Profiles.....	31
The Identification Profile.....	31
Deleting Consumer Applications.....	34
Registering Application Assets as Consumers of an Asset.....	35
Deploying or Publishing Consumer Applications to Gateways.....	38
Overview.....	38
Deploying Consumer Applications Using CentraSite Control UI.....	38
Publishing Consumer Applications Using CentraSite Business UI.....	39
Publishing Consumer Applications from the Search Results Page.....	40
Publishing a Consumer Application from the Actions Bar.....	40
Deploying Consumer Applications from the Command Line.....	40

Write a Script File to Invoke Deployment Operation (for Windows).....	40
Write a Script File to Invoke Deployment Operation (for UNIX).....	41
Synchronizing Consumer Applications in Mediator from the Command Line.....	42
Viewing the Deployment History Log Using CentraSite Control UI.....	42
Consumer Registrations.....	45
Introduction to Consumer Registration.....	46
Scope of Consumer Registration at Design-Time.....	47
Scope of Consumer Registration at Run-Time.....	47
The Concept of Registering a Consumer for an Asset.....	47
Permissions Needed to Register Consumers.....	48
The Design/Change-Time Policy Used for Consumer Registration.....	48
Implementing Consumer Registration.....	49
Overview.....	49
Design-Time Consumer Registration Scenario.....	49
Run-Time Consumer Registration Scenarios.....	50
Scenario A: Virtual Service without API Consumption Settings and Evaluate Policy Actions.....	51
Scenario B: Virtual Service without API Consumption Settings and with Evaluate Policy Actions.....	52
Scenario C: Virtual Service with API Consumption Settings and without Evaluate Policy Action.....	52
Scenario D: Virtual Service with API Consumption Settings and Evaluate Policy Action.....	54
Viewing Consumer Registration Requests.....	55
Monitoring Consumer Count of an Asset.....	55
Viewing or Modifying Details of a Consumer.....	56
Unregistering an Existing Consumer.....	56
Unregistering an Existing Consumer from the Consumed Asset Details Page.....	57
Unregistering an Existing Consumer from the Consumer Asset Details Page.....	58
Run-Time Gateways.....	59
Introduction to Gateways.....	60
Who Can Create and Manage Gateways?.....	60
Creating and Managing Gateways.....	61
Overview.....	61
Creating a Gateway Asset.....	61
Configuring Gateway Details.....	65
Deleting a Gateway Instance.....	65
Virtualized Services in CentraSite.....	67
Introduction to Virtualized Services.....	68
The Predefined Asset Types Installed with CentraSite for Virtualization.....	69
Who Can Create and Manage Virtual Services?.....	69
Creating Virtual Services.....	70
Creating a Virtual Service from Scratch.....	71

Virtualizing an Existing Service.....	72
Configuring SOAP-based Virtual Services.....	74
The Entry Protocol Step (SOAP).....	74
The Request Processing Step (SOAP).....	76
The Response Processing Step (SOAP).....	78
The Routing Protocols Step (SOAP).....	82
“Straight Through” Routing (SOAP).....	82
“Content-based” Routing (SOAP).....	87
Creating a Routing Rule for “Content-based” Routing (SOAP).....	91
“Context-based” Routing (SOAP).....	93
Creating a Routing Rule for “Context-based” Routing (SOAP).....	97
“Load Balancing” Routing (SOAP).....	100
The Routing Protocol for Services Exposed over JMS (SOAP).....	106
Viewing REST/XML-based Virtual Services.....	107
The Entry Protocol Step (REST/XML).....	108
The Request Processing Step (REST/XML).....	108
The Response Processing Step (REST/XML).....	109
The Routing Protocols Step (REST/XML).....	109
The “Straight Through” Routing Protocol Step (REST/XML).....	109
The “Content-based” Routing Protocol Step (REST or XML).....	112
The “Context-based” Routing Protocol Step (REST or XML).....	115
The “Load Balancing” Routing Protocol Step (REST or XML).....	118
Viewing or Editing Virtualized Services.....	120
The Summary Profile.....	122
The Technical Details Profile.....	122
The Specification Profile.....	123
The Consumers Profile.....	124
The Permissions Profile.....	124
The Policies Profile.....	125
The Deployment Profile.....	125
The Performance Profile.....	125
The Events Profile.....	126
Revising Virtualized Services.....	127
Creating Run-Time Policies.....	128
Actions that Run-Time Policies can Execute.....	128
Who Can Create and Manage Run-Time Policies?.....	130
Creating a Run-Time Policy.....	130
Setting Permissions on a Run-Time Policy.....	134
Who Can Set Permissions on a Run-Time Policy?.....	134
Assigning Permissions to a Run-Time Policy.....	134
Activating a Run-Time Policy.....	135
Deactivating a Run-Time Policy.....	137
Viewing the Run-Time Policy List.....	137
Modifying a Run-Time Policy.....	139
Viewing or Changing a Run-Time Policy.....	139

Modifying Actions for a Run-Time Policy.....	141
Modifying the Actions List.....	141
Modifying Action Parameters.....	142
Modifying the Scope of a Run-Time Action.....	143
Viewing the List of Services To Which a Run-Time Policy Applies.....	144
Deleting a Run-Time Policy.....	144
System-Assigned vs. User-Assigned Version Identifiers.....	145
Asymmetric Binding Configuration.....	145
Asymmetric Binding.....	146
The Asymmetric Binding Components.....	147
Initiator Token Inclusion.....	147
Recipient Token Inclusion.....	147
Algorithm Suite.....	147
Layout.....	148
The Asymmetric Binding Configuration Command Tool.....	148
get Asymmetric Binding.....	149
set Asymmetric Binding.....	149
remove Asymmetric Binding.....	150
Deploying and Undeploying Virtualized Services to Targets.....	151
The Synchronous Deployment Model.....	151
Who Can Deploy Virtualized Services to Targets?.....	153
Conditions that Must be Satisfied for Effective Deployment of Virtualized Services.....	153
What Happens When You Deploy a Virtualized Service?.....	154
Deploying, Undeploying, and Redeploying Virtualized Services.....	155
Deploying Virtualized Services from a Service's Detail Page.....	156
Deploying Virtualized Services from the Operations > Deployment Page.....	158
Selecting a Target and Services to be Deployed on the Selected Target.....	161
Using a Keyword Search to Select a Target or Service to Deploy.....	161
Using an Advanced Search to Select a Target or Service to Deploy.....	163
Deploying Virtualized Services Using Command Line Tool.....	164
Deploy a Virtualized Service to Mediator.....	164
Undeploy a Virtualized Service from Mediator.....	165
Deploy, Undeploy or Redeploy Multiple Virtualized Services in Mediator.....	166
Configuring API Key Header Using Command Line Tool.....	166
Deploying Virtualized Services Using a Batch Process.....	167
Viewing the Deployment History Log.....	168
Deleting a Deployment Activity Log.....	170
Performing a Delete Operation Using the Deployed Assets Tab.....	170
Performing a Delete Operation Using the Deployment History Tab.....	171
Securing Communications with for Synchronous Deployment.....	171
Anatomy of a SSL Connection.....	172
SSL Connection Type.....	172
As an SSL Client.....	172
As an SSL Server.....	173
Roadmap for Configuring SSL.....	173

Creating Keys and Certificates.....	174
Creating a Keystore and Truststore.....	175
Obtaining the Certificates and Keys of the webMethods Mediator.....	175
Keystores and Truststores.....	175
Keystore File.....	175
Truststore File.....	176
How Uses a Keystore and Truststore.....	176
Protecting Keystore and Truststore Files.....	177
Configuring CentraSite to Use SSL.....	177
Configure CentraSite Client to Use One-way SSL.....	178
Configure CentraSite Client to Use Two-way SSL.....	179
Using the CTP Server.xml File for SSL.....	181
Configuring webMethods Integration Server to Use SSL.....	182
Configure Integration Server to Use One-way SSL.....	182
Configure Integration Server to Use Two-way SSL.....	183
Configuring webMethods Mediator to Use SSL.....	185
Virtualized APIs in CentraSite Business UI.....	187
Introduction to Virtualized APIs.....	188
Roles and Permissions Needed to Create and Manage Virtualized APIs.....	188
Preparing to Virtualize and Publish the Virtualized APIs.....	189
Run-Time Components of Virtualized API.....	190
Creating Virtualized APIs.....	203
Before You Begin.....	203
Ways in Which You Can Create Virtualized APIs.....	203
Creating a Virtualized API from an Existing Native API.....	204
Creating a Virtual SOAP API.....	204
Add a Virtual SOAP API.....	204
Assign Run-Time Actions for the Virtual SOAP API.....	205
Implement Virtualization and Publish the Virtual SOAP API to Gateways.....	206
Creating a Virtual REST API.....	206
Add a Virtual REST API.....	206
Assign Run-Time Actions for the Virtual REST API.....	207
Implement Virtualization and Publish the Virtual REST API to Gateways.....	207
Creating a Virtualized API from Scratch.....	207
Creating a Virtual SOAP API.....	207
Add a Virtual SOAP API.....	207
Upload an Input File.....	209
Assign Run-Time Actions to the Virtual SOAP API.....	209
Publish the Virtual SOAP API to Gateways.....	210
Creating a Virtual REST API.....	210
Add a Virtual REST API.....	210
Add Resources and Methods.....	211
Assign Run-Time Actions to the Virtual REST API.....	211
Publish the Virtual REST API to Gateways.....	211

Reconfiguring a Virtualized API.....	211
Ways in Which You Can Reconfigure Virtualized APIs.....	212
Reconfiguring a Virtualized API from the Native API Details Page.....	212
Reconfiguring a Virtualized API from the Virtualized API Details Page.....	213
Resource Synchronization in Virtual REST APIs.....	214
Resource Synchronization Usage Scenarios.....	216
Edit Resource Usage Scenario.....	217
Add Resource Usage Scenario.....	218
Delete Resource Usage Scenario.....	218
Combination of Usage Scenarios.....	218
Versioning Virtual API or Service.....	219
Creating a Version for Virtual API or Service.....	220
Publishing a Versioned Virtual API or Service.....	221
Runtime Versioning Support in Mediator.....	222
Viewing or Changing a Virtualized API.....	223
Viewing the Virtualized API Specific Profiles.....	224
Identification Profile.....	225
Identification Profile (for Assets with Key-based Authentication).....	227
OAuth2 Identification Details Profile (for Assets with OAuth-based Authentication).....	227
API Key Scope Profile.....	228
Assigning Run-Time Actions to a Virtualized API.....	228
Before You Begin.....	228
Ways in Which You Assign Run-Time Actions to Virtualized APIs.....	230
Modifying the Action List.....	230
Configuring Policy Action Parameters.....	231
Publishing and Unpublishing APIs to and from Gateways.....	231
The Process of Publishing an API to Mediator.....	232
The Process of Publishing an API to API-Portal.....	232
Roles and Permissions Needed to Publish and Unpublish APIs.....	232
Important Considerations When Publishing an API.....	233
Use Cases for Publishing an API.....	234
Use Case A: Publish an API From the Native API Details Page.....	234
Use Case B: Publish an API From the Virtualized API Details Page.....	235
Use Case C: Publish an API from the Search Results Page.....	235
Publishing API(s) to Gateway(s).....	238
Publishing an Individual API from the Native API Details Page.....	239
Publishing an Individual API from the Virtualized API Details Page.....	241
Publishing Multiple APIs from the Search Results Page.....	242
Unpublishing API(s) from Gateway(s).....	242
Unpublishing an Individual API from the Native API Details Page.....	242
Unpublishing an Individual API from the Virtualized API Details Page.....	243
Unpublishing Multiple APIs from the Search Results Page.....	244
Exposing a Virtual SOAP API as Virtual REST API.....	244
Displaying Runtime Information for a Virtualized API.....	245

The Runtime Metrics.....	245
Displaying the Runtime Metrics.....	246
The Runtime Events.....	247
Displaying the Runtime Events.....	248
Obtaining Your API Keys and Access Tokens for Consumption.....	250
Fetching and Using Your API Keys for Consumption.....	250
The API Consumption Model.....	250
How Does Mediator Evaluate Consumers at Run Time?.....	251
How Does a Consumer Use the Generated API Key?.....	252
Request Header.....	252
Query String.....	252
What Happens When You Request for API Consumption?.....	252
What Happens When Consumption Fails?.....	252
Fetching and Using Your OAuth2 Access Tokens for Consumption.....	253
Ways for Clients to Provide the Inputs.....	253
Using HTTPS for Granting Access Tokens.....	254
Responses Returned to Clients.....	254
Managing Your API Keys.....	254
Fetching Details of Your API Keys.....	255
Viewing Details of an API Key Using the Email Notification.....	255
Viewing Details of an API Key Using the API Details Page.....	256
Viewing Details of an API Key Using the User Preferences.....	256
Renewing Your API Key.....	257
Revoking Your API Key.....	258
Deleting Your API Key.....	259
Deleting an Individual API key.....	259
Deleting Multiple API Keys in a Single Operation.....	259
Privileged User of a Virtualized API.....	260
Invoking webMethods IS Services in Virtual Services.....	261
Introduction.....	262
Using the Security API in webMethods IS Services.....	263
pub.mediator.security.ws:AddUsernameToken.....	263
pub.mediator.security.ws:AddX509Token.....	265
pub.mediator.security.ws:AddSamlSenderVouchesToken.....	266
pub.mediator.security.ws:AddTimestamp.....	268
pub.mediator.addressing:AddWSAddressingHeaders.....	270
Using Context Variables in Virtual Services.....	273
Introduction to Context Variables in Virtual Services.....	274
The Predefined Context Variables.....	274
The API for Context Variables.....	279
pub.mediator.ctxvar:getContextVariable.....	279
pub.mediator.ctxvar:setContextVariable.....	281
pub.mediator.ctxvar:declareContextVariable.....	282
pub.mediator.ctxvar:removeContextVariable.....	283

Sample Flow Service: Getting a Context Variable Value.....	284
Sample Flow Service: Setting a Context Variable Value.....	288
Important Considerations when Configuring SOAP-based Virtual Services.....	295
Handling Services with Multiple Ports and Bindings.....	296
Workaround Option 1.....	297
Workaround Option 2.....	298
Important Considerations when Configuring REST or XML Virtual Services.....	301
Introduction.....	302
Endpoint Manipulation of REST or XML Virtual Services.....	302
Example 1.....	303
Example 2.....	303
Example 3.....	303
Example 4.....	303
Example 5.....	304
Example 6.....	304
The Request Message's HTTP Methods and Content-Types for REST and XML Services...	304
Changing the HTTP Method of a REST or XML Request.....	305
The Implications of Changing HTTP Methods.....	306
Changing HTTP Methods in Requests Dynamically using a Context Variable.....	308
Sample XSLT Transformation for GET-to-POST or GET-to-PUT.....	309
Working with the JSON Content-Type.....	310
How Mediator Determines Which Builder and Formatter Classes to Use (and How You Can Override Them).....	311
Scenarios for Requesting JSON Type Services.....	313
JSON Example 1: GET Request, JSON Response.....	315
JSON Example 2: POST/JSON Request, JSON Response (where POST is not supported).....	316
JSON Example 3: GET Request, XML Response.....	318
Characteristics of the Mapped and Badgerfish JSON Conventions.....	319
Mapped JSON Convention.....	319
Badgerfish Convention.....	320
Multiple Root Nodes in JSON REST Services.....	320
Handling Virtual REST APIs with Multiple Resources.....	321
Scenario A.....	322
Scenario B.....	322
Straight-Through Routing Action.....	323
Workaround Option 1:.....	323
Workaround Option 2:.....	324
Content-Based Routing Action.....	324
Context-Based Routing Action.....	325
Run-Time Governance Reference.....	327
Run-Time Events and Key Performance Indicator (KPI) Metrics.....	328
The Run-Time Event Types.....	328

The Key Performance Indicator (KPI) Metrics.....	329
The Event Notification Destinations.....	330
Destinations for the Monitoring and Transaction Events.....	330
SMTP Email Servers.....	331
The Integration Server's Local Log.....	331
The Integration Server's Audit Log.....	332
The Metrics Tracking Interval.....	332
Configuring CentraSite to Receive Run-Time Events and Metrics.....	332
Components of the Event Receiver.....	333
Configuring the Event Receiver.....	334
Setting the Database Configuration Properties.....	334
Setting the SNMPv3 Transport Configuration Properties.....	335
Setting the SNMPv3 USM Configuration Properties.....	336
Setting the Events Queue Implementation Property.....	337
Setting the Properties for FileSystem or InMemory.....	338
Event Type Modeling.....	339
The "Target Type to Event Type Association" Object.....	341
Event Modeling.....	342
Viewing Run-Time Events and Metrics.....	343
Viewing Run-Time Events and Metrics for Targets.....	343
Viewing Run-Time Events and Metrics for Virtual Services.....	345
Viewing Run-Time Events and Metrics for APIs.....	345
Creating Custom Run-Time Events.....	345
Modifying Run-Time Events.....	346
Built-In Run-Time Actions Reference for Virtual Services.....	347
Summary of the Run-Time Actions for Virtual Services.....	347
WS-SecurityPolicy 1.2 Actions.....	347
Authentication Actions (WS-SecurityPolicy 1.2).....	347
XML Security Actions (WS-SecurityPolicy 1.2).....	347
Monitoring Actions.....	348
Additional Actions.....	348
Configuring Destinations for Alerts and Logs.....	349
The watt.server.auth.skipForMediator Property.....	350
Action Evaluation Order and Dependencies.....	351
Effective Policies.....	351
Usage Cases for Identifying/Authenticating Consumers.....	355
Run-Time Actions Reference for Virtual Services.....	356
Authorize Against Registered Consumers.....	357
Authorize User.....	357
Identify Consumer.....	358
Log Invocation.....	360
Monitor Service Performance.....	363
Monitor Service Level Agreement.....	366
Require Encryption.....	371
Require HTTP Basic Authentication.....	373

Require Signing.....	374
Require SSL.....	375
Require Timestamps.....	376
Require WSS SAML Token.....	376
Require WSS Username Token.....	377
Require WSS X.509 Token.....	378
Throttling Traffic Optimization.....	378
Validate Schema.....	381
Built-In Run-Time Actions Reference for APIs.....	382
Summary of the Run-Time Actions.....	382
Request Handling Actions.....	382
Policy Enforcement Actions.....	383
Authentication Actions.....	383
JMS Routing Actions.....	384
Logging and Monitoring Actions.....	384
Routing Actions.....	385
Security Actions.....	385
Traffic Management Action.....	388
Validation Action.....	388
Response Handling Actions.....	388
Error Handling Action.....	389
The watt.server.auth.skipForMediator Property.....	389
Effective Policies.....	390
Usage Cases for Identifying/Authenticating Clients.....	401
Run-Time Actions Reference.....	402
Allow Anonymous Usage.....	402
Content Based Routing.....	403
Context Based Routing.....	407
Conditional Error Processing.....	411
Failure Messages.....	415
Enable REST Support.....	417
Evaluate Client Certificate for SSL Connectivity.....	417
Evaluate Hostname.....	419
Evaluate HTTP Basic Authentication.....	419
Evaluate IP Address.....	421
Evaluate Kerberos Token.....	422
Evaluate OAuth2 Token.....	423
Evaluate WSS Username Token.....	424
Evaluate WSS X.509 Certificate.....	426
Evaluate XPath Expression.....	427
HTTP Basic Authentication.....	428
Invoke webMethods Integration Server.....	430
JMS Routing Rule.....	430
Load Balancing and Failover Routing.....	433
Log Invocation.....	437

Monitor Service Level Agreement.....	439
Monitor Service Performance.....	443
NTLM Authentication.....	446
OAuth2 Authentication.....	448
Response Transformation.....	449
Request Transformation.....	450
Require Encryption.....	451
Require HTTP / HTTPS.....	453
Require JMS.....	454
Require Signing.....	455
Require SSL.....	457
Require Timestamps.....	457
Require WSS SAML Token.....	458
Set Custom Headers.....	459
Set JMS Headers.....	459
Set Media Type.....	461
Set Message Properties.....	461
Straight Through Routing.....	462
Throttling Traffic Optimization.....	466
Service Result Cache.....	469
Validate Schema.....	472
Computed Runtime Actions.....	472
Writing Your Own Computed Runtime Action.....	473
The Build Environment.....	473
Implementation Guidelines for Computed Runtime Action.....	473
Implementation for Computed Action UI.....	474
Implementation for Computed Action Parser.....	475
Setting up the Computed Action Plug-in.....	477
Activating the Computed Action.....	478
Sample Computed Runtime Action.....	478

About this Guide

This guide describes the basic components of CentraSite's run-time governance environment. Additionally, it describes how you can use the CentraSite graphical user interfaces to design and configure your run-time governance environment.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention <i>folder.subfolder:service</i> .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
<i>Italic</i>	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 Overview of Run-Time Governance

- Components of the Run-Time Governance Environment 18

Components of the Run-Time Governance Environment

When you use CentraSite to govern Web services at run time, the basic components in the run-time environment are:

- A policy enforcement point such as webMethods Mediator.

webMethods Mediator is a service mediation and policy enforcement application for Web services. The Web services can be SOAP-based, REST-based or plain XML Web services.

Mediator hosts virtual services, which are proxy services that receive requests from consumers on behalf of particular Web services. Mediator can also host virtualized APIs, which are proxy APIs for Web services that have been externalized as Application Programming Interfaces (APIs).

Mediator enforces the run-time governance policies or rules that you define for your virtual services or virtualized APIs (such as security enforcement and audit-trail logging) and handles mediation measures between consumer and provider (such as message transformation and message routing).

Besides serving as an intermediary between consumer applications and native services, Mediator also collects performance statistics and event information about the traffic flowing between consumers and the native services, and reports this data to CentraSite.

Mediator is designed for use with CentraSite. The Mediator application is delivered as a package called WmMediator, which runs on webMethods Integration Server. It provides an infrastructure for the run-time governance policies or rules that you define.

- Virtualized services or virtualized APIs.

You can choose to expose your Web services as virtualized services. When you create a virtualized service, you also define:

- Run-time governance policies for the virtualized service.
- The consumer applications that are used to allow consumers to access the virtualized service.

As an alternative to exposing your Web services as virtualized services, you can instead expose the Web services as virtualized APIs, which perform a role similar to virtualized services. When you virtualize an API, you also define:

- Policy enforcement rules for the API.
- API keys, which enable users to securely access the API.

You define all these components in CentraSite, and you store and manage them from CentraSite's UDDI registry/repository.

- Native services, which are Web services that process requests submitted by consumers. If a native service produces a response, it returns the response to the virtualized service or virtualized API, and the virtualized service or virtualized API returns it to the consumer.
- CentraSite, which serves several key roles in the run-time environment. Besides serving as the system of record for the artifacts in the SOA environment (such as virtualized services or virtualized APIs and their run-time governance policies or rules), CentraSite provides the tools you use to define these artifacts and deploy them to Mediator. Additionally, CentraSite receives and logs the performance metrics and event data collected by Mediator and provides tools for viewing this data.

Subsequent sections of the CentraSite documentation provide the information you need for designing and configuring your run-time governance environment in CentraSite. The following sub-sections provide a brief overview of those subsequent sections:

Run-Time Gateways

To use an instance of CentraSite with Mediator, you must define a run-time gateway that identifies the specific instance of Mediator that you want to use. A run-time gateway is a registry object that represents a particular instance of a policy enforcement point (in this case, an instance of webMethods Mediator). The gateway object specifies the address of the Mediator's deployment endpoint, which is the endpoint that CentraSite uses to interact with Mediator to deploy virtualized services or virtualized APIs.

If you use multiple Mediators with an instance of CentraSite, you must create a gateway for each Mediator. To make the Mediators easier to distinguish when they are viewed in CentraSite, consider adopting a naming convention for gateways that clearly identifies to which environment the gateway belongs (for example, development, test, production). You can deploy any given virtualized service or virtualized API to one or more run-time gateways.

Instead of (or in addition to) using webMethods Mediator for mediation and/or policy enforcement, you can use other third-party products with CentraSite. Support for third-party policy-enforcement and run-time governance tools is available through integrations that are provided by members of the CentraSite Community. These tools are made available through the [Software AG TECHcommunity Website](#).

In addition to using Mediator, you can use webMethods Insight. Insight is an additional monitoring tool from Software AG that you can use with CentraSite. Insight enables you to see what is happening in real-time with service transactions as they flow across any system. It provides visibility and control at the transaction level to heterogeneous SOA environments. CentraSite provides support for Insight as a gateway type out-of-the-box. For more information about Insight's uses and capabilities, see the Insight user documentation.

For detailed information about run-time gateways, see "[Run-Time Gateways](#)" on page 19.

Virtualized Services in CentraSite Control

If you choose to expose Web services as virtualized services, you need to:

- **Create the virtualized services.** A virtualized service is a service that runs on Mediator and acts as the consumer-facing proxy for a native service that runs elsewhere on the network. You can create virtualized services for Web services that are SOAP-based, REST-based or plain XML Web services. A virtualized service provides a layer of abstraction between the service consumer and the service provider, and promotes loose coupling by providing independence of location, protocol and format between the consuming application and the provider service. You create virtualized services using the CentraSite Control user interface, and store them in the CentraSite registry/repository.
- **Create run-time governance policies for the virtualized services.** A run-time governance policy is a sequence of actions that are carried out by a policy-enforcement point (such as Mediator) when a consumer requests a particular Web service through the policy-enforcement point. The actions in a run-time policy perform activities such as identifying/authenticating consumers, validating digital signatures and capturing performance metrics. You create run-time policies using the CentraSite Control user interface, and store them in the CentraSite registry/repository.
- **Create consumer applications, which specify the consumers that are allowed to consume services and other assets at run time.** A consumer application is a computer application that consumes (invokes) assets (services, BPEL processes or XML schemas) at run time. Typically, you create a consumer application to specify the consumers that are allowed to consume a particular asset. Then, you include the consumer application in the Consumers profile of the asset.

A consumer application defines the precise consumer identifiers (for example, a list of user names in HTTP headers, a range of IP addresses, and so on). Thus the policy enforcement point (such as Mediator) can identify or authenticate the consumers that are requesting an asset. You create consumer applications using the CentraSite Control user interface, and store them in the CentraSite registry/repository.

- **Deploy the virtualized services and consumer applications to Mediator.**

For detail information about virtualized services in CentraSite Control, see "[Virtualized Services in CentraSite Control](#)" on page 20.

Virtualizing APIs Using the CentraSite Business UI

As an alternative to exposing your Web services as virtualized services, you can instead expose the Web services as virtualized APIs.

Like virtualized services, virtualized APIs run on Mediator and act as the consumer-facing proxy for a native service that runs elsewhere on the network. You can create a virtualized API for Web services that are SOAP-based, REST-based or plain XML Web services.

CentraSite provides an Application Programming Interface (API) Management platform that you can implement to expose Web services as virtualized APIs.

CentraSite's Application Programming Interface (API) Management platform enables enterprises to selectively externalize their new and existing assets as APIs across various channels, monitor the interface's lifecycle with an integrated infrastructure, and make sure the needs of developers and application using the API are met.

Application Program Interfaces (APIs) are the new distribution channel for CentraSite assets. With an integrated infrastructure, you can:

- Securely expose your APIs to external developers and partners (that is, any external entities with which your enterprise interacts, such as suppliers and other vendors, dealers and distributors, customers, government agencies, trade organizations, and so forth).
- Provide design time and run time governance capabilities to the APIs.

To support this new distribution channel, CentraSite's API Management enables developers, architects, and business developers to:

- Publish the right APIs into their organization's central registry.
- Discover APIs and use them to assemble new applications.
- Manage the entire process of creating, publishing, deploying, and retiring APIs.
- Obtain detailed information about an API, including: the list of its consumers, its technical support contacts, its disposition in the development lifecycle, usage tips, and performance data.
- Control access to CentraSite and to the metadata for individual APIs listed in the registry.
- Impose mandatory approval processes to ensure that APIs accepted into the SOA adhere to organizational standards and policies.
- Get notifications on the APIs they use.
- Model the lifecycle process associated with each API and specify the events that are to be triggered when an API transitions from one lifecycle state to another.

For detailed information about virtualizing APIs using the CentraSite UI, see *Working with the CentraSite Business UI* and ["Creating Virtualized APIs" on page 203](#).

Run-Time Governance Reference Information

This guide provides details about:

- The run-time events and Key Performance Indicator (KPI) metrics that can be collected and reported for each virtualized service or virtualized API deployed in your system.

This section also describes how to configure CentraSite to receive the events and metrics from the policy-enforcement point (such as Mediator) that collects them.

- The run-time actions for virtualized services. (There is a separate set of run-time actions for virtualized APIs.)

You use these actions only when you are using CentraSite Control to create run-time policies for virtualized services. This section provides:

- An alphabetic reference of all actions and their parameters.
- A listing of the action evaluation order and action dependencies.
- Some common combinations of actions used to authenticate/identify consumers.

- The run-time actions for virtualized APIs.

You use these actions only when you are using the CentraSite Business UI to create policy enforcement rules for virtualized APIs. These actions are similar in functionality to the actions for virtualized services. This section provides:

- An alphabetic reference of all actions and their parameters.
- A listing of the action evaluation order and action dependencies.

For details, see "[Run-Time Governance Reference Information](#)" on page 21.

2 Consumer Applications

■ Introduction to Consumer Applications	24
■ Roles and Permissions Needed to Create and Manage Consumer Applications	24
■ Creating and Managing Consumer Applications	25
■ Registering Application Assets as Consumers of an Asset	35
■ Deploying or Publishing Consumer Applications to Gateways	38

Introduction to Consumer Applications

A consumer application is a computer application that consumes (invokes) assets (Web services, XML services or REST services) at run time. Typically, you create consumer applications to specify the consumers that are allowed to consume a particular asset. Then, you include the consumer application in the **Consume Asset** dialog of the asset.

A consumer application in CentraSite is represented by an *Application* asset. The Application asset defines the precise consumer identifiers (for example, a list of user names in HTTP headers, a range of IP addresses, and so on). Thus the policy enforcement point can identify or authenticate the consumers that are requesting an asset.

You can use Application assets with any supported policy enforcement point (that is, webMethods Mediator or any supported third-party policy enforcement point).

Note: If you want to authenticate consumers (using LDAP or another external authentication mechanism), make sure that your policy enforcement point is configured to enable authentication. For information, see the documentation for your policy enforcement point.

Roles and Permissions Needed to Create and Manage Consumer Applications

To create and manage (view, edit, and delete) consumer applications, you must have the following roles or permissions:

- CentraSite Administrator
- Organization Administrator
- Asset Provider
- Instance level Full permission for Application asset
- If you have the CentraSite Administrator role, you can create and manage consumer applications within any organization.
- If you have the Organization Administrator role or API-Portal Administrator role for a specific organization, you have the ability to create and manage consumer applications within that specific organization.

In addition to the above, you can also assign the following permissions to a user for a consumer application:

- Instance level view only
- Instance level modify permission

- Instance level full permission

For more information about roles and permissions, see *Getting Started with CentraSite*.

Creating and Managing Consumer Applications

Overview

You can create, update, list, and delete consumer applications in CentraSite using the following user interfaces:

- CentraSite Control UI
- CentraSite Business UI

Adding and Managing Consumer Applications Using CentraSite Control UI

Adding a Consumer Application

Use the following procedure to add a consumer application and save it to the CentraSite registry/repository.

To add a consumer application asset

1. In CentraSite Control, go to **Asset Catalog > Browse**.
2. Click **Add Asset(s)**.
3. In the **Add Asset** dialog, complete the following fields:

<u>In this field...</u>	<u>Specify...</u>
Type	The Application asset type.
Name	A name for the application asset. An asset name can contain any character (including spaces).
Description	<i>Optional.</i> A comment or descriptive information about the new application asset.
Organization	The organization to which the application asset belongs.
Initial Version	<i>Optional.</i> An identifier for the initial version of the application asset. The default is 1.0, but you can use any versioning scheme

In this field...	Specify...
	<p>you choose. The version identifier does not need to be numeric. Examples:</p> <pre data-bbox="527 388 1386 493">0.0a 1.0.0 (beta) Pre-release 001 v1-2007.04.30</pre> <p>You can later create new versions of the application asset (see <i>CentraSite User's Guide</i>).</p>

4. Click **OK**.
5. Configure the asset's extended attributes as described in the subsequent sections.

Modifying Consumer Application Details

If you want to modify the information stored for a consumer application, proceed as follows:

To modify the details stored for a consumer application

1. In CentraSite Control, go to **Asset Catalog > Browse**.
2. On the Browse page, select the check box for **Application** type, and then click **Update**.
3. Right-click an application and click **Details**, or select the check boxes for multiple applications, click the **Actions** menu, and click **Details**.
4. Modify the application's basic information such as: Name, Description or user-defined Version number.
5. If you want to modify the application's additional information, select the individual profile and modify the fields as necessary.
6. When you have finished making your edits, click **Save**.

Configuring Consumer Application Profiles

Use the following procedure to configure the profile details of a consumer application.

The Identification Profile

In this profile, specify the precise values for the consumer identifier(s) that you specified in the **Identify Consumer** action.

- Note:** Keep the following in mind:
- If you specify *multiple* identifiers, the system evaluates them with the identifier defined in the **Identify Consumer** action.
 - If you want to authenticate consumers, make sure that your policy enforcement point is configured to enable authentication. For information,

see the webMethods Mediator documentation or the documentation for your third-party PEP.

To configure the Identification profile

- Specify values for one or more of the following fields.

Note: The value(s) that you specify in the **Identification** profile depend on how the run-time policy's Identify Consumer is configured. For example, if **Identify Consumer** is configured to identify consumers by their IP address, you should specify the consumer IP addresses here. For information about this action, see the "[Built-In Run-Time Actions Reference for Virtual Services](#)" on page 347.

In this field...	Do the following...
IPv4 Address	<p>Identify consumers based on their originating 4-byte IP address.</p> <p>Use this field when the Identify Consumer action is configured to identify consumer applications by IP address.</p> <ul style="list-style-type: none"> ■ To specify an individual IP address, type the address in the From field. The application asset will identify only those requests that originate from the specified IP address. ■ To specify a range of IP addresses, type the lowest IP address in the From field and the highest IP address in the To field. For example, the values 192.168.0.0 and 192.168.0.10 indicates that requests originating from any IP address that lies between the specified range will be identified by the application asset. <p>If you need to specify additional IP addresses, use the plus button to add more rows.</p>
IPv6 Address	<p>As for IPv4 Address, but using the 128-bit IPv6 format. For example: 1234:5678:9ABC:DEF0:1234:5678:9ABC:DEF0.</p>
Identification Token	<p>Identify consumers based on one or more of the following kinds of identification tokens:</p> <p>Use this field when the Identify Consumer action is configured to identify consumer applications by host name, HTTP user name, WSS user name or a custom token.</p> <ul style="list-style-type: none"> ■ Host Name—To identify consumers based on a specified host name, type the host name (for example, pcmachine.ab.com) in the Name field. The application asset

In this field...	Do the following...
	<p>will identify only those requests that originate from the specified host name.</p> <ul style="list-style-type: none"> <li data-bbox="581 394 1412 804">■ HTTP Authentication Token—To authenticate consumers based on the user name that is transmitted in an HTTP authentication user token, type the user name (for example, testuser123) in the Name field. The application asset will identify only the requests that contain the specified user name encoded and passed in the HTTP authentication user token. Authentication is handled by LDAP or another external authentication mechanism. You can specify the kinds of HTTP headers that Mediator will pass from requests to services. The default is the Authorization header. To configure Mediator to pass other kinds of HTTP headers, see the Mediator documentation. <li data-bbox="581 825 1412 1098">■ WS-Security Authentication Token—To authenticate consumers based on the user name that is transmitted in the SOAP or XML message header (HTTP body), type the user name (for example, userwss) in the Name field. The application asset will identify only the requests that contain the specified user name passed in the SOAP or XML message header. Authentication is handled by LDAP or another external authentication mechanism. <li data-bbox="581 1119 1412 1421">■ Custom identification token (XPath)—To identify consumers based on the result of applying an XPath expression on the SOAP or XML message or request, enter the XPath expression in the Name field. For example, typing <code>//*[local-name()='Envelope']/*[local-name()='Body']/*[local-name()='echoInt']/*[local-name()='echoIntInput='][.='2']</code> in the Name field will identify the requests that contain the XPath and the consumers.

If you need to specify additional tokens, use the plus button to add more rows.

Consumer Certificate

Identify consumers based on information in an X.509 v3 certificate.

Use this field when the **Identify Consumer** action is configured to identify consumer applications by a consumer certificate.

Click **Browse** to locate the certificate (.cer) file and select the certificate file.

The Permissions Profile

Use this profile to set permissions for the application. For information, see the *CentraSite User's Guide*.

The Versions Profile

Use this profile to generate a new version of the application. For information, see the *CentraSite User's Guide*.

The Subscriptions Profile

Use this profile to view the list of users that are registered to receive notifications when changes are made to the application.

If you want to be notified whenever an application is updated, click the **Actions** button, and select **Notify me** from the drop-down list. Your user name will appear in the **Subscriptions** profile. Any other users who have permission to access the application can add their own user names to this list.

The Audit Log Profile

Use this profile to view an audit log of the recent changes to the application (including changes in an asset's lifecycle state).

The Object-Specific Properties Profile

Use this profile to add object-specific properties to the application.

If you want to configure this profile, click the **Add Property** button, and complete the fields as necessary.

Deleting Consumer Applications

Before you delete a consumer application, we strongly recommend that you examine the application's dependency usecase using the Asset Navigator in CentraSite Business UI to determine whether other assets will be affected by the application's deletion. To visualize the dependencies for a consumer application using the Asset Navigator, see *Working with the CentraSite Business UI*.

To delete consumer applications

1. In CentraSite Control, go to **Asset Catalog > Browse**.
2. On the Browse page, select the check box for **Application** type, and then click **Update**.
3. Select the check box for one application and click **Delete**, or select the check boxes for multiple applications, click the **Actions** menu, and click **Delete**.

Adding and Managing Consumer Applications Using CentraSite Business UI

Creating a Consumer Application

Use the following procedure to create a consumer application and save it to the CentraSite registry/repository.

Note: Creating a new consumer application from scratch using **Create Assets** wizard is explained in this section. You can also create a consumer application asset on-the-fly from the details page of the asset you want to consume using the **Consume Asset** dialog. Refer to the section "[Implementing Consumer Registration](#)" on page 49 for details on the **Create a new Application asset** link.

To create a consumer application asset

1. In the activity bar, click **Create Assets**.
When you do this, you see the **Create Asset** wizard.
2. In the **Basic Information** section, complete the following fields:

In this field...	Specify...
Name	A name for the application asset. An asset name can contain any character (including spaces).
Type	The Application asset type.
Organization	The organization to which the application asset belongs.
Initial Version	<p><i>Optional.</i> An identifier for the initial version of the application asset. The default is 1.0, but you can use any versioning scheme you choose. The version identifier does not need to be numeric. Examples:</p> <pre>0.0a 1.0.0 (beta) Pre-release 001 V1-2007.04.30</pre>
Description	<p><i>Optional.</i> A comment or descriptive information about the new application asset.</p>

3. Click **Next**.

Note: If you do not specify value for a required field, CentraSite issues an error icon. Clicking the red-colored icon displays an error description.

4. In the **Preview** section, review the information, and click **Finish**.
5. Configure the application's extended attributes as described in the subsequent sections.

Modifying Consumer Application Details

If you want to modify the information stored for a consumer application, proceed as follows:

To modify the details stored for a consumer application

1. Display the list of consumer applications. For instructions, see *Working with the CentraSite Business UI*.
2. In the displayed list, click the link of the application whose details you want to view. This shows the details of the application.

The details include:

- The application's basic information (version number, the asset type, owning organization, the current lifecycle state, the last modified date, owning user, the total number of watchers, consumers and pending approvals (if any), and a general description of the application).
 - Additional information about the application.
3. If you want to modify the application's details, click the **Edit** icon. You can then enter new values for the application's fields.
 4. Modify the application's basic information such as: Name, Description or user-defined Version number.
 5. If you want to modify the application's additional information, select the individual profile and modify the fields as necessary.
 6. When you have finished making your edits, click **Save**.

Configuring Consumer Application Profiles

Use the following procedure to configure the profile details of a consumer application.

The Identification Profile

In this profile, specify the precise values for the consumer identifier(s) that you specified in the **Evaluate** * actions.

Note: Keep the following in mind:

- If you specify *multiple* identifiers, the system evaluates them with the identifier defined in the **Evaluate** * actions.

- If you want to authenticate consumers, make sure that your policy enforcement point is configured to enable authentication. For information, see the webMethods Mediator documentation or the documentation for your third-party PEP.

To configure the consumer identifiers

- Specify values for one or more consumer identifier tokens.

Note: The value(s) that you specify in the Identification profile depend on how the run-time policy's **Evaluate** * actions are configured. For example, if an **Evaluate IP Address** action is configured to identify and validate consumers by their IP address, you should specify the consumer IP addresses here. For information about the **Evaluate** * actions, see "[Built-In Run-Time Actions Reference for APIs](#)" on page 382. .

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

In this field...

Do the following...

Identification Token

Identify and authenticate consumers based on one or more of the following kinds of identification tokens:

Use this field when the **Evaluate** * action is configured to identify and authenticate consumer applications by host name, HTTP user name, WSS user name or a custom token.

- **Host Name**—To identify consumers based on a specified host name, type the host name (for example, pcmachine.ab.com) in the Name field. The application asset will identify only those requests that originate from the specified host name.
- **HTTP Authentication Token**—To identify and authenticate consumers based on the user name that is transmitted in an HTTP authentication user token, type the user name (for example, testuser123) in the Name field. The application asset will identify only the requests that contain the specified user name encoded and passed in the HTTP authentication user token. Authentication is handled by LDAP or another external authentication mechanism. You can specify the kinds of HTTP headers that Mediator will pass from requests to services. The default is the Authorization header. To configure Mediator to pass other kinds of HTTP headers, see the Mediator documentation.

In this field...	Do the following...
	<ul style="list-style-type: none"> <li data-bbox="605 323 1404 600">■ WS-Security Authentication Token—To identify and authenticate consumers based on the user name that is transmitted in the SOAP or XML message header (HTTP body), type the user name (for example, userwss) in the Name field. The application asset will identify only the requests that contain the specified user name passed in the SOAP or XML message header. Authentication is handled by LDAP or another external authentication mechanism. <li data-bbox="605 621 1404 926">■ Custom identification token (XPath)—To identify consumers based on the result of applying an XPath expression on the SOAP or XML message or request, enter the XPath expression in the Name field. For example, typing <code>//*[local-name()='Envelope']/*[local-name()='Body']/*[local-name()='echoInt']/*[local-name()='echoIntInput=''] [.='2']</code> in the Name field will identify the requests that contain the XPath and the consumers. <p data-bbox="605 947 1404 1010">If you need to specify additional tokens, use the plus button to add more rows.</p>
<p data-bbox="285 1058 574 1089">From IP-V4 Address</p>	<p data-bbox="605 1058 1404 1121">Identify consumers based on their originating 4-byte IP address.</p> <p data-bbox="605 1142 1404 1247">Use this field when the Evaluate IP Address action is configured to identify consumer applications based on their originating IP addresses.</p> <p data-bbox="605 1268 1404 1402">To specify an individual IP address, type the address in the From IP-V4 Address field. The application asset will identify only those requests that originate from the specified IP address. Example: <code>192.168.0.0</code></p>
<p data-bbox="285 1451 574 1482">To IP-V4 Address</p>	<p data-bbox="605 1451 1404 1482">Identify consumers based on their 4-byte IP address range.</p> <p data-bbox="605 1503 1404 1608">Use this field when the Evaluate IP Address action is configured to identify consumer applications based on their 4-byte IP address range.</p> <p data-bbox="605 1629 1404 1829">To specify a range of IP addresses, type the lowest IP address in the From IP-V4 Address field and the highest IP address in the To IP-V4 Address field. For example, the values <code>192.168.0.0</code> and <code>192.168.0.10</code> indicates that requests originating from any IP address that lies between the specified range will be identified by the application asset.</p>

In this field...	Do the following...
From IP-V6 Address	<p>Identify consumers based on their originating 6-byte IP address.</p> <p>Use this field when the Evaluate IP Address action is configured to identify consumer applications based on their originating IP addresses.</p> <p>As for IPv4 Address, but using the 128-bit IPv6 format. Example: 1234:5678:9ABC:DEF0:1234:5678:9ABC:DEF0</p>
To IP-V6 Address	<p>Identify consumers based on their 6-byte IP address range.</p> <p>Use this field when the Evaluate IP Address action is configured to identify consumer applications based on their 6-byte IP address range.</p> <p>As for IPv4 Address, but using the 128-bit IPv6 format. For example: 1234:5678:9ABC:DEF0:1234:5678:9ABC:DEF0</p>
Consumer Certificate	<p>Specify the X.509 certificates to identify requests from the specified consumer.</p> <p>Click Upload, and select the certificate (.cer) file.</p>
Partner ID	<p>Specifies the trading partner ID.</p> <p>When a consumer application is identified in Mediator and the event logging is enabled, you can locate the partner ID in the event data of the identified consumer. You can leverage the event data for partner based analytics.</p> <p>Note: The Partner ID attribute is introduced for integration scenarios based on webMethods Trading Networks.</p>

Deleting Consumer Applications

Before you delete a consumer application, we strongly recommend that you examine the application's **Asset Navigator** action (using the Business UI) to determine whether other assets will be affected by the application's deletion.

To delete consumer application

1. Display the list of consumer applications. For instructions, see *Working with the CentraSite Business UI*.
2. Select the check box for one application and click **Delete**, or select the check boxes for multiple applications, and click **Delete**.

Registering Application Assets as Consumers of an Asset

Note: Beginning with version 9.9, CentraSite does not support registering Application assets as consumers of an asset using the Control UI. This means that there is no consumer registration procedure in Control UI. As a result:

- The **Register as Consumer** action in the details page of an Application asset is removed in CentraSite Control user interface.
- You cannot modify details of the consumers using the **Consumers** profile of an Application asset in the CentraSite Control user interface. You can only view details of the consumers such as: Name, Description, Type of Consumer, Owner, Organization, Created date, Modified date, and the Lifecycle State.
- However, you can still deploy, undeploy, and redeploy consumer applications to Mediator gateway in CentraSite Control user interface.

Therefore, you cannot register Application assets as consumers of an asset using the CentraSite Control UI. Instead, you can use the enhanced interface of CentraSite Business UI which supports consumer registration for users, groups, Application and arbitrary assets (in contrast, earlier versions of CentraSite Business UI supported a standardized interface for consumer registration of Application assets only). Documentation of the prior consumer registration interface is available to CentraSite customers who have a current maintenance contract in Empower Product Support website.

To register Application assets as consumers of an API using the Business user interface, you perform the following high-level steps:

1. **Include the Evaluate (consumer) action in the API's run-time policy.**

To identify the consumer applications that are requesting an API, that asset must have a run-time message flow that includes one or more **Evaluate *** actions. In these actions, you specify the consumer identifier(s) you want to use for identifying consumer applications. (Alternatively, you may configure these actions to allow unrestricted access.) These actions extract the specified identifier from an incoming request and locate the consumer application defined by that identifier.

For example, if you configure the **Evaluate IP Address** action to identify consumers by a specific IP address, the PEP extracts the IP address from a request's HTTP header at run time and searches its list of Application assets for the application that is defined by that specified IP address.

You can configure the **Evaluate *** actions to identify consumer applications based on one or more of the following consumer identifiers in a request message:

Action	Consumer Identifier	Description
Evaluate IP Address	IP Address	The IP address from which the request message to call the API had originated.
Evaluate Hostname	Identification Token - Host Name	The name of the host machine from which the request message to call the API had originated.
Evaluate HTTP Basic Authentication	Identification Token - HTTP Authentication Token	The user ID submitted by the requestor when it was asked to provide basic HTTP credentials (user name and password).
Evaluate WSS Username Token	Identification Token - WS-Security Authentication Token	The WSS username token supplied in the header of the SOAP or XML request message to call the API.
Evaluate XPath Expression	Identification Token - Custom Identification	A string produced by applying a specified XPath expression to the SOAP or XML request message to call the API.
Evaluate WSS X.509 Certificate	Consumer Certificate	The X.509 certificate supplied in the header of the SOAP or XML request message to call the API.
Evaluate Client Certificate for SSL Connectivity	Consumer Certificate - Client Certificate for SSL Connectivity	The client's certificate that the consumer application submits to the asset. The certificate is supplied during the SSL handshake over the Transport layer. Communication between the client and the asset must be over HTTPS.
Evaluate OAuth2 Token	Identification Token - OAuth2 Token	The client's OAuth2 token supplied in the SOAP or XML request message to call the API.

When deciding which **Evaluate *** action to use to identify and authenticate a consumer application, consider the following points:

- Whatever identifier you choose to identify a consumer application, it must be unique to the application. Identifiers that represent user names are often not suitable because the identified users might submit requests for multiple applications.
- Identifying applications by IP address or host name is often a suitable choice, however, it does create a dependency on the network infrastructure. If a consumer application moves to a new machine, or its IP address changes, you must update the identifiers in the Application asset.
- Using X.509 certificates or a custom token that is extracted from the SOAP or XML message itself (using an XPATH expression), is often the most trouble-free way to identify a consumer application.

For information about the **Evaluate** * actions, see ["Built-In Run-Time Actions Reference for APIs" on page 382](#).

2. **Create an Application asset in the registry.**

In the Application asset you specify precise values for the consumer identifier(s) that you specified in the **Evaluate** * actions. For details, see ["The Identification Profile" on page 31](#).

3. **Specify the Application asset in the Consume Asset dialog of the API you want to consume.**

Use the **Consume** action in the details page of the API you want to consume. This opens the **Consume Asset** dialog. For procedures, see ["Implementing Consumer Registration" on page 49](#)

The run-time behavior of identifying and authenticating consumers is as follows:

1. CentraSite translates the Application asset to the appropriate WS-Security policy actions or an equivalent XML when the Application asset is enforced by the PEP.
2. When a consumer application requests access to an API, the PEP tries to map the consumer's identifier (which is found in the request) to an identifier in the Application asset.
 - If the identifier is an IP address, a host name, a custom identification string or a consumer certificate, the PEP tries to identify the consumer (the consumer is not authenticated).
 - If the identifier is an HTTP Authentication token or a WS-Security Authentication token, the PEP tries to *authenticate* the consumer. If you use webMethods Mediator, authentication is handled by LDAP or by another external authentication mechanism, depending on how Mediator is configured. If you use a third-party PEP, authentication capabilities depend on the PEP.

3. The identified or authenticated consumer information is published back to the registry as part of the transaction or other events. This information is used to correlate the consumer-specific run-time dependencies.

Deploying or Publishing Consumer Applications to Gateways

Overview

You can deploy/publish consumer applications to a policy enforcement point (PEP) such as webMethods Mediator using the following two methods:

- CentraSite Control UI
- CentraSite Business UI
- Command Line Interface

Deploying Consumer Applications Using CentraSite Control UI

The following procedure describes how to deploy consumer applications to a Mediator target using the CentraSite Control user interface.

To deploy consumer applications using CentraSite Control UI

1. In CentraSite Control, go to **Operations > Deployment**.
2. On the Deploy Consumers tab, click **Synchronize**.
3. In the **Select a Target and Consumer Applications to be Deployed** dialog box, perform a keyword or advanced search to display the list of consumer applications and targets that are ready for deployment.
 - If you want to perform a keyword search and you need procedures for this step, see ["Using a Keyword Search to Select a Target or Service to Deploy"](#) on page 161.
 - If you want to perform an advanced search and you need procedures for this step, see ["Using a Advanced Search to Select a Target or Service to Deploy"](#) on page 163.
4. Click **OK**.
5. The deployment process is carried out by a synchronous mechanism between CentraSite and Mediator:
 - a. CentraSite invokes the Mediator's deployer service and pushes the consumer applications that are ready for deployment to the Mediator.
 - b. Instantly, Mediator deploys the consumer applications that were received from CentraSite and notifies CentraSite when the deployment process is complete.

The Deploy Consumers tab includes the following information:

Column	Description						
Pending Changes	Icons indicating the deployment status of the consumer applications.						
	<table border="1"> <thead> <tr> <th>Icon</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td></td> <td>The consumer application is deployed to the target.</td> </tr> <tr> <td></td> <td>The consumer application is pending deployment to the target.</td> </tr> </tbody> </table>	Icon	Description		The consumer application is deployed to the target.		The consumer application is pending deployment to the target.
Icon	Description						
	The consumer application is deployed to the target.						
	The consumer application is pending deployment to the target.						
Rule ID	The synchronization rule ID of the target (that is, webMethods Mediator or Insight).						
Target	The name of the target on which the consumer application is deployed.						
Last Sync Date	The date/time that the deployment occurred.						
Status	The deployment status of the consumer application (e.g., Deployed or Failed).						

Important: If the status shown in the **Status** column does not automatically switch to **Deployed** or **Failed**, click the **Refresh** button to determine whether CentraSite was able to deploy the consumer applications successfully. If the deployment process failed, identify and correct the error and then try deploying the consumer application again.

Publishing Consumer Applications Using CentraSite Business UI

The following procedure describes how to publish consumer applications to a Mediator gateway using the CentraSite Business user interface.

Note: CentraSite Business UI does not have a dedicated user interface for publishing consumer applications to Mediator. However, if you are publishing an API with an associated consumer application to Mediator, the consumer application information is also sent to Mediator. Consider you have a Virtual REST API with an Application asset as consumer, when you publish the Virtual REST API to Mediator, the consumer application details are also sent.

Publishing Consumer Applications from the Search Results Page

To publish consumer applications from the Search Results page

1. Display the list of consumer applications. For instructions, see *Working with the CentraSite Business UI*.
2. Select the check box for one application and click **Publish**, or select the check boxes for multiple applications, and click **Publish**.
3. Complete the fields. For instructions, see ["Publishing and Unpublishing APIs to and from Gateways" on page 231](#)
4. Click **Publish**.

Publishing a Consumer Application from the Actions Bar

To publish a consumer application from the actions bar

1. Display the list of consumer applications. For instructions, see *Working with the CentraSite Business UI*.
2. In the displayed list, click the name of the consumer application you want to publish. For instructions, see *Working with the CentraSite Business UI*.
3. In the actions bar for the consumer application, click **Publish**.
This opens a dialog for publishing the consumer application to the selected Mediator gateways.
4. Complete the fields. For instructions, see ["Publishing and Unpublishing APIs to and from Gateways" on page 231](#)
5. Click **Publish**.

Deploying Consumer Applications from the Command Line

The deployment operation of a consumer application invoked between CentraSite and Mediator using command line, includes a `main()` method, which allows it to be called from a Windows batch file or from a UNIX shell script.

Write a Script File to Invoke Deployment Operation (for Windows)

Create a script file that looks as follows if CentraSite is running under Windows.

```
@echo off
set JAVAEXE=fullPathToJava.exe set REDIST=CentraSiteHomeDirectory\redist
set BASEDIR=%~dp0
cd /d %REDIST%

REM build CLASSPATH with all files from jar directory
set LOCAL_CLASSPATH=
for %%I in (*.jar) do call "CentraSiteHomeDirectory\bin\cfg\lcp.cmd" %%I

%JAVAEXE% -cp %LOCAL_CLASSPATH% deployerClassName %*
cd /d %BASEDIR%
```

Where *deployerClassName* is the name of the deployer that you want to run.

Example

The following is an example of a script file that calls the Consumer Applications deployer:

```
@echo off
REM
REM Run Consumer Applications deployer
REM
set JAVAEXE=D:\software\java\jdk1.5.0_12\bin\java
set REDIST=C:\SoftwareAG\CentraSite\redist
set BASEDIR=%~dp0
cd /d %REDIST%

REM build CLASSPATH with all files from jar directory
set LOCAL_CLASSPATH=
for %%I in (*.jar) do call "C:\SoftwareAG\CentraSite\bin\cfg\lcp.cmd" %%I

%JAVAEXE% -cp %LOCAL_CLASSPATH%
com.softwareag.centrasite.runtime.pep.util.VirtualServiceDeployer %*
cd /d %BASEDIR%
```

Write a Script File to Invoke Deployment Operation (for UNIX)

Create a script file that looks as follows if CentraSite is running under Unix.

```
set javaexe="fullPathToJava.exe"
set redist="CentraSiteHomeDirectory/redist"
set mainjar="CentraSiteRuntimePEP.jar"
set delim='\:'
cd "$redist"
set cl=""
foreach j ( `ls *.jar` )
  if ($cl != "") set cl=${cl}${delim}
  set cl=${cl}${j}
end
setenv CLASSPATH ${mainjar}${delim}${cl}
$javaexe deployerClassName $*
```

Where *deployerClassName* is the name of the deployer that you want to run.

Example

The following is an example of a script file that calls the Consumer Applications deployer:

```
#!/bin/csh
#
# Run Consumer Applications deployer
#
set javaexe="/mydir/softwareag/cjp/v16/bin/java"
set redist="/mydir/softwareag/CentraSite/redist"
set mainjar="CentraSiteRuntimePEP.jar"
set delim='\:'
# build CLASSPATH with all files from jar directory
cd "$redist"
set cl=""
foreach j ( `ls *.jar` )
  if ($cl != "") set cl=${cl}${delim}
```

```

set cl=${cl}${j}
end
setenv CLASSPATH ${mainjar}${delim}${cl}
$javaexe com.softwareag.centrasite.runtime.pep.util.VirtualServiceDeployer $*

```

Synchronizing Consumer Applications in Mediator from the Command Line

To synchronize consumer applications in Mediator, use a `sync consumers` command of the following format:

```

C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd sync consumers
[-url <CENTRASITE-URL>] -user <USER-ID> -password <PASSWORD> -gateway
<GATEWAY>

```

Input Parameters

The following table describes the complete set of input parameters that you can use with the `sync consumers` command:

Input Parameter	Description
<code>-url</code>	The fully qualified URL (<code>http://localhost:53307/CentraSite/CentraSite</code>) for the CentraSite registry/repository.
<code>-user</code>	The user ID of a user who has the CentraSite Administrator role.
<code>-password</code>	The password of the user identified by the parameter <code>-user</code> .
<code>-gateway</code>	The gateway on which to synchronize the consumer applications.

For example (all in one line):

```

C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd sync consumers
-url http://localhost:53307/CentraSite/CentraSite -user Administrator -
password manage -gateway Gateway1

```

Viewing the Deployment History Log Using CentraSite Control UI

The Deployment History Log contains information about the consumer applications that CentraSite has pushed to the Mediator for deployment.

To view the deployment history log in CentraSite, you must belong to an Administrator role.

- If your user account belongs to the CentraSite Administrator role, you can view the deployment history log entries for consumer applications that belong to any organization.

- If your user account belongs to the Organization Administrator or Operations Administrator role for an organization, you can view the deployment history log entries for consumer applications that belong to that organization.
- If you do not belong to either of these roles, but you have the Mediator Publisher role, you can view the deployment history log entries for consumer applications that you created.

To view the Deployment History Log

1. In CentraSite Control, go to **Operations > Deployment**.
2. Select the **Deployment History** tab.

When you do this, you see a list of all of the consumer applications that are recently sent to Mediator.

3. If you want to filter the list, type a partial string in the **Search** field. CentraSite applies the filter to the **Name** column. The **Search** field is a type-ahead field, so as soon as you enter any characters, the display will be updated to show only those consumer applications whose name contains the specified characters. The wildcard character “%” is supported.

The **Deployment History** tab provides the following information about each application.

Column	Description
Rule ID	The synchronization rule-ID that CentraSite automatically generates up on creation of the Mediator in CentraSite Control.
Name	The name assigned to the consumer application.
Version	The user-assigned version identifier for the consumer application.
Target Name	The name of the target on which the consumer application is deployed.
Action	The deployment action of the consumer application on the Mediator. Possible Values: Deployed, Undeployed.
Status	The deployment status of the consumer application on the Mediator Possible Values: Success, Failed.
Date	The date/time that the consumer application has deployed, redeployed or undeployed.

4. To view details of a particular deployment workflow, click the hyperlinked value in the **Name** column.

3 Consumer Registrations

■ Introduction to Consumer Registration	46
■ The Concept of Registering a Consumer for an Asset	47
■ Permissions Needed to Register Consumers	48
■ The Design/Change-Time Policy Used for Consumer Registration	48
■ Implementing Consumer Registration	49
■ Viewing Consumer Registration Requests	55
■ Monitoring Consumer Count of an Asset	55
■ Viewing or Modifying Details of a Consumer	56
■ Unregistering an Existing Consumer	56

Introduction to Consumer Registration

Note: Beginning with version 9.9, CentraSite does not support the Consumer Registration functionality in Control UI. This means that there is no consumer registration procedure in Control UI. As a result:

- The **Register as Consumer** action in the details page of an Application asset is removed in CentraSite Control user interface.
- You cannot modify details of the consumers using the **Consumers** profile of an Application asset in the CentraSite Control user interface. You can only view details of the consumers such as: Name, Description, Type of Consumer, Owner, Organization, Created date, Modified date, and the Lifecycle State.
- The **Consumer Registrations** section which provides functionality for viewing a summary of all consumer registration requests in conjunction with **Pending Registrations** and **Registration Requests** is removed from the **Home > My CentraSite > Inbox** page of CentraSite Control user interface.
- The **My Pending Consumer Registration Requests** data feed that was used to construct a custom portlet for rendering the list of all consumer registration requests in CentraSite Business user interface is deprecated and will be removed in future releases.

Therefore, you cannot register users, or groups, or Application assets as consumers of an asset using the CentraSite Control UI. Instead, you can use the enhanced interface of CentraSite Business UI which supports consumer registration for users, groups and arbitrary assets (in contrast, earlier versions of CentraSite Business UI supported a standardized interface for consumer registration of Application assets only). Documentation of the prior consumer registration interface is available to CentraSite customers who have a current maintenance contract in Empower Product Support website.

The term “*consumer registration*” means providing users the ability to consume assets. Consumer registration with CentraSite enables asset providers to establish an enhanced level of protection and access control as to who can consume the assets using configurable approval workflows and thus allows the asset providers to visualize and control the consumption of their assets.

CentraSite's flexible and extensible asset catalog enables the asset providers to expose their reusable assets. At any time, asset consumers can discover the reusable assets in the asset catalog which functions as the central registry, and reuse them in their own applications. Consumer registration functionality provides the mechanism to establish the consumer-provider relationships in CentraSite at both design-time and run-time.

Scope of Consumer Registration at Design-Time

At design-time, this functionality lets you:

- Register any arbitrary asset as consumer of other assets in your organization.
- Fetch details about all of the consumers of a specific asset, also details about all of the assets that a specific asset is consuming. To generate a report data on the list of consumers, see
- Visualize the consumption relationship that exists between asset providers and asset consumers. This helps you to identify the artifacts in the registry that will be affected if an asset is not available or must be changed. To visualize the dependencies between assets using the Asset Navigator, see *Working with the CentraSite Business UI*.

Scope of Consumer Registration at Run-Time

At run-time, this functionality lets you:

- Enforce a level of run-time security checks by allowing a consumer to specify details about the consuming applications that in turn will be used for identification of that application during invocation of the asset at run-time.
- Determine whether a particular asset consumer is authorized to invoke the target asset.
- Access the asset's metadata (that is, view additional profiles) and/or receive notifications when changes occur to an asset that they consume.

The Concept of Registering a Consumer for an Asset

Consumer registration in CentraSite has two major elements: it has an asset, which denotes the asset to be consumed; and it has a consumer asset, which denotes the asset consuming the asset. In the CentraSite Business user interface, an asset that is consumed by another asset is represented by the **Consumed Assets** attribute, and the consuming asset is represented by the **Consumers** attribute in the details page of an asset.

CentraSite users with View permissions for assets can provide a consumer from any of the following four categories:

- **A registered user of CentraSite** - CentraSite users can register themselves as consumers of the specified assets. That is, users access the asset's metadata (that is, view additional profiles) and/or receive notifications when changes occur to an asset that they consume.
- **A registered group of users** - CentraSite users can register a specific set of users who are represented by a group as consumers of the specified assets.
- **A consumer who is identified by an asset instance of Application type** - CentraSite users can register Application assets as consumers of the specified assets. The Application

assets contain precise characteristics by which Mediator can identify messages from these specified Application assets at run time.

- **A consumer who is identified by an arbitrary asset instance of the consuming type** - CentraSite users can register arbitrary assets as consumers of the specified assets. These are asset instances whose types are defined to be as the consumers of the specified assets in their type definition.

In general, an asset type definition specifies a default set of consuming types, for example, Users, Groups, and Applications, whose instances are allowed to consume instances of the specified type. An administrator can optionally modify this type definition and remove the default consumption type to disable the consumer registration for all assets of the default type removed from the type definition. For example, an administrator might want to remove the default Application type in the BPEL type definition to disable the Application assets from consuming any asset of the type BPEL.

Permissions Needed to Register Consumers

Any user with a **View** instance-level permission on the asset to be consumed can register consumers for that asset. In addition to registering consumers for a particular asset, a user can also create new assets and register the newly created assets as consumer of that asset, provided that the user has Create Assets permission on the organization to which the asset belongs.

The Design/Change-Time Policy Used for Consumer Registration

The enhanced consumer registration functionality enables the consumer registration process without explicitly creating a consumer-registration policy, and also completes the consumer registration process without requiring the owner of the asset to review and accept the user's request.

If you want to impose an approval process on consumer registration, that is, control which consumers can invoke an asset, you can create a design-time policy with one of the CentraSite's built-in approval actions for the **OnConsumerRegistration** event. For example, you can create a design-time policy, named Consumer-Registration Policy, which requires a designated group of approvers to review and approve all consumer requests for invoking the asset. Upon reviewing, the approvers may approve or reject such requests.

For more information about creating the Consumer-Registration Policy, refer to the section "Using Approvals with OnConsumerRegistration Events" in the *CentraSite User's Guide*.

When you register as consumers of an asset, the policy is triggered and the "Register as Consumer" request is submitted to all members of the approval list specified in the Initiate Approval action. Then, the approvers can either approve or decline the request.

If the approvers approve the request, the consumers will be registered as consumers for the asset. Consider, you have configured the **Set Consumer Permission** action in this policy; CentraSite assigns the appropriate permissions to the specified set of users, groups and arbitrary assets.

Implementing Consumer Registration

Overview

CentraSite offers various scenarios for exercising the consumer registration at design-time and run-time in Business user interface. At design-time, you might want to register as consumer of an asset instance of any asset type, for example, “REST service” in which case you do not have to specify any authentication settings (API Key or OAuth Token) or consumer identifiers, and would access the asset to examine the usage; whereas at run-time, you might want to register as consumer of an asset instance of a virtual type, for example, “Virtual REST Service” in which you case you will have to specify the authentication settings or consumer identifiers or both, and would invoke the asset to query and obtain the data.

Design-Time Consumer Registration Scenario

Pre-requisite: To impose an approval process, ensure that you have created a design-time policy, for example, the Consumer-Registration Policy, as described in the previous section.

To register as a consumer of an asset of this scenario, proceed as follows:

1. Display the details page of the asset you want to consume. If you need procedures for this step, see *Working with the CentraSite Business UI* for details.
2. In the actions bar for the asset, click the **Consume** icon. This opens the **Consume Asset** dialog for registering as a consumer.
3. If you want to register an existing asset as consumer, do one of the following:
 - Enter the name of the asset in the textbox. Then click the plus button next to the textbox or press Enter to add the asset to the list of currently selected assets.
 - Click **Choose**. In the **Choose Consume Assets** dialog, do the following:
 - i. Enter the name of the asset that you want to register as a consumer in the textbox. Then click the **Search** icon.
 - ii. CentraSite populates a set of instances whose asset types are defined to be consumers to instances of this asset type. For more details on the consumable definition, see the *CentraSite Administrator's Guide*.
 - iii. In the displayed list, mark the checkbox of the asset that you want to register. If you want to select more than one asset, mark the checkbox of each asset that you want to register. Then click **OK** to complete the selection.

4. If you want to create and register a new asset with this workflow, do the following:
 - a. Click the **Create a new asset** link. This opens the Create New Asset page.
 - b. Choose the asset type for which an asset will be created. The **Types** drop-down list reflects its behavior in terms of the type definition for the displayed asset. The list will contain the list of top-level asset types which you have defined as consuming type in the asset type definition.

For example, if you have defined Applications, Users, Groups, and XML Schemas as consumers in the REST Service type definition, when trying to create a new asset using the **Consume** action of a displayed REST service, the **Types** drop-down contains only asset types, namely, Applications and XML Schemas, which are the top-level asset types defined as consuming types of the REST service.

If none of the asset types, for example, Users, Groups, and a Custom Type, defined as consuming types in the type definition of a displayed REST service is a top-level type, the **Create a new asset** link will not appear in the **Consume Asset** dialog.
 - c. After you specify the value for all of the required attributes, click **Save** to save the new asset. The details page for the asset that you just created is displayed.
 - d. Once again display the details page of the asset you want to consume, and then click the **Consume** action. In the **Consume Asset** dialog, enter the name of the newly created asset, click **Search**.
5. When you do this, you see a list of all of the currently selected assets.
6. If at any time you wish to remove an asset that is currently selected for consumption, move the mouse over the asset you want to remove. Click the icon to remove the asset.
7. In the **Reason for Request** field, enter the reason for requesting consumption of the asset.

This field will be visible to all designated approvers who would review the consumer registration request for the asset.
8. Click **Consume**.

Run-Time Consumer Registration Scenarios

Pre-requisite: To impose an approval process, ensure that you have created a design-time, for example, the Consumer-Registration Policy, as described in the previous section.

An API provider (owner of the API) specifies the type of authentication (API key or OAuth2 token using the consumption settings) and configures a set of Evaluate * actions (using the run-time policies) to identify and authorize the consumer that request the consumption access to the metadata of the particular API. Based on the specified enforcement definitions for the API, CentraSite offers interactive user interfaces for the following four exemplary scenarios:

Is API Consumption Settings Configured?	Is an Evaluate Action Configured?	For Procedures...
No	No	See, "Scenario A" on page 51
No	Yes	See, "Scenario B" on page 52
Yes	No	See, "Scenario C" on page 52
Yes	Yes	See, "Scenario D" on page 54

Scenario A: Virtual Service without API Consumption Settings and Evaluate Policy Actions

Prerequisites for scenario A are:

- *The asset should be an instance of the virtual type - Virtual Service, Virtual REST Service, Virtual XML Service.*
- *The API consumption settings or an Evaluate * policy action should not be configured for the asset.*

To register as a consumer of an asset of this scenario, proceed as follows:

1. Display the details page for the asset to which you want to register as a consumer. If you need procedures for this step, see *Working with the CentraSite Business UI* for details.
2. In the actions bar for the asset, click **Consume**.
3. In the **Consume Asset** dialog, you can choose to do one of the following:
 - Register an existing asset as consumer of the displayed asset.
 - Create a new asset in CentraSite, and then register the newly created asset as consumer of this asset.

When you register as a consumer of an asset instance of virtual type, which does not include the consumption settings or evaluate actions, the fields displayed for consumption are the same as for the asset instance of base type, so for a description of the fields, follow the instructions provided above for registering as a consumer at design-time scenario.

Scenario B: Virtual Service without API Consumption Settings and with Evaluate Policy Actions

Prerequisites for scenario B are:

- *The asset should be an instance of the virtual type* - Virtual Service, Virtual REST Service, Virtual XML Service.
- The API consumption settings *should not be configured* for the asset.
- There should be at least one Evaluate policy action included within the asset's runtime configuration.

To register as a consumer of an asset of this scenario, proceed as follows:

1. Display the details page for the asset to which you want to register as a consumer. If you need procedures for this step, see *Working with the CentraSite Business UI* for details.
2. In the actions bar for the asset, click **Consume**.
3. In the **Consume Asset** dialog, you can choose to do one of the following:
 - Register an existing asset as consumer of the displayed asset. For instructions, see ["Design-Time Consumer Registration Scenario" on page 49](#).
 - Create a new Application asset on-the-fly, and then register the newly created Application asset as consumer of this asset. Proceed as follows:
 - i. Click the **Create a new Application asset** link. This opens a dialog in which you can enter the required information for the Application asset.
 - ii. Enter the appropriate information for each of the displayed data fields.

Note: The list of identifiers is dynamically loaded depending upon the Evaluate policy actions that are defined for the asset.
 - iii. Click **Create** to add the new Application asset to CentraSite.
4. When you do this, you see a list of all of the currently selected assets.
5. If at any time you wish to remove an asset that is currently selected for consumption, move the mouse over the asset you want to remove. Click the icon to remove the asset.
6. Click **Consume**.

Scenario C: Virtual Service with API Consumption Settings and without Evaluate Policy Action

Pre-requisites for scenario C are:

- *The asset should be an instance of the virtual type* - Virtual Service, Virtual REST Service, Virtual XML Service.

- The API consumption settings *should be configured* for the asset.
- An Evaluate * policy action *should not be included* in the asset's run-time configuration.

To register as a consumer of an asset of this scenario, proceed as follows:

1. Display the details page for the asset to which you want to register as a consumer. If you need procedures for this step, see *Working with the CentraSite Business UI* for details.
2. In the actions bar for the asset, click **Consume**.
3. In the **Consume API** dialog, you can choose to do one of the following:
 - Generate access tokens for an existing Consumer Application asset to make API calls on the displayed asset. To do so, follow the steps below:
 - i. Select the type of authentication token that you want to use to allow the consumer to access the asset.
 - **API Key** - CentraSite will generate an API key (a base64-encoded string of the consumer-key:consumer-secret combination) to access and test the asset.
 - **OAuth2 Token** - CentraSite will generate the OAuth2 client credentials (a client_id and client_secret) to further request an OAuth2 token to access and test the asset.

For more details about the usage of the authentication tokens, see "[Obtaining Your API Keys and Access Tokens for Consumption](#)" on page 250.

For more details on the configuration settings for API Key/OAuth2 authentication, refer to the section "Configuring the API Consumption Settings" in *Working with the CentraSite Business UI*.

- ii. Enter the name of the Consumer Application asset in the field labeled **Consumer Application Name**.
 - iii. Mark the **Email me** checkbox to automatically generate an email notification about the usage of access token to the user at the email address provided at registration.
 - iv. Enter reason of the request in the textbox.
 - v. Click **Consume**.
- Register an arbitrary asset as consumer of the asset. To do so, follow the steps below:
 - i. Click on the **Consume using other assets** link.
 - ii. In the **Consume Asset** dialog, you can choose to do one of the following:
 - Register an existing asset as consumer of the displayed asset.

- Create a new asset in CentraSite, and then register the newly created asset as consumer of this asset.

For instructions, see "[Design-Time Consumer Registration Scenario](#)" on page 49.

Scenario D: Virtual Service with API Consumption Settings and Evaluate Policy Action

Pre-requisites for scenario D are:

- *The asset should be an instance of the virtual type - Virtual Service, Virtual REST Service, Virtual XML Service.*
- *The API consumption settings should be configured for the asset.*
- *An Evaluate * policy action should be included in the asset's run-time configuration.*

To register as a consumer of an asset of this scenario, proceed as follows:

1. Display the details page for the asset to which you want to register as a consumer. If you need procedures for this step, see *Working with the CentraSite Business UI* for details.
2. In the actions bar for the asset, click **Consume**.
3. In the **Consume Asset** dialog, you can choose to do one of the following:
 - Register an existing Application asset as consumer and generate access tokens to make API calls on the displayed asset.
 - i. Complete the fields. The fields displayed are the same fields as for the Scenario C, so for a description of the fields, follow the instructions provided above for "[Virtual Service with API Consumption Settings and without Evaluate Policy Action](#)" on page 52.
 - ii. Enter the appropriate information for each of the displayed consumer identifier fields. CentraSite automatically populates the list of identifiers depending on the set of Evaluate policy actions that are defined for the displayed asset. For more information about each identifier, see "[Configuring Consumer Application Profiles](#)" on page 26.
 - Register an arbitrary asset as consumer of the asset. To do so, follow the steps below:
 - i. Click on the **Consume using other assets** link.
 - ii. In the **Consume Asset** dialog, you can choose to do one of the following:
 - Register an existing asset as consumer of the displayed asset.
 - Create a new Application asset on-the-fly, and then register the newly created Application asset as consumer of this asset.

For instructions, see "[Design-Time Consumer Registration Scenario](#)" on page 49.

Viewing Consumer Registration Requests

Note: Beginning with version 9.9, CentraSite no longer supports the concept of consumer registration requests. This means that there is no procedure for approving the pending consumer registration requests, and reviewing the status of consumer registration requests in both CentraSite Control and Business user interfaces.

The **Consumer Registrations** section which provides functionality for viewing a summary of all consumer registration requests in conjunction with **Pending Registrations** and **Registration Requests** is removed from the **Home > My CentraSite > Inbox** page of CentraSite Control user interface.

The **My Pending Consumer Registration Requests** data feed that was used to construct a custom portlet for rendering the list of all consumer registration requests in CentraSite Business user interface is deprecated and will be removed in future releases.

However, if for example, you have an approval process initiated by a consumer registration policy, and you are a designated approver, CentraSite places the consumer registration requests for your review and approval. You can review and accept these requests on the **My Pending Approvals** portlet of your Welcome page in CentraSite Business user interface. Alternatively, you can review and accept these requests on the **Pending Approvals** tab of your Inbox page in CentraSite Control user interface.

If you have made a register as a consumer request for an asset, on successful registration, you will see the Consumers count incremented by one in the asset details page.

Important: If you migrate CentraSite from a pre-9.9 version, the consumer registration requests that were pending for an approval by the asset owner in the previous version of CentraSite will continue to remain in the new CentraSite Registry Repository. To fetch details of these migrated consumer registration requests, you must execute the `list Pending Consumer Registrations` command in the command line interface of CentraSite. Refer to the section "Fetching Details of the Migrated Pending Consumer Registrations from the Command Line" in the *CentraSite Administrator's Guide* for details on the command line tool.

Monitoring Consumer Count of an Asset

CentraSite helps monitor the registered consumers for each displayed asset.

The number of consumers is typically represented by a numerical prefix to the attribute **Consumers** in the description area of the Basic Information profile in the asset details page. For example, if you have 5 users registered as consumers for the displayed asset, then the **Basic Information** profile displays **5 Consumers**. If you do not have a registered consumer, then this attribute displays **0 Consumers**.

To see a list of the currently registered consumers for the displayed asset, click on the consumer count. In the displayed list, click the link of the consumer whose details you want to view. This shows the details of the consumer.

In a similar way, you can monitor the number of assets consumed by a particular consumer using the attribute **Consumed Assets** in the description area of the **Basic Information** profile in the consumer asset details page. For example, if you have registered as a consumer of 5 other assets in the registry, then the **Basic Information** profile displays **5 Consumed Assets**. If you are not consuming any other asset, then this attribute displays **0 Consumed Assets**.

Viewing or Modifying Details of a Consumer

If you want to examine or modify the details stored for a consumer, proceed as follows:

1. Display the details page of the asset whose consumers you want to view or modify.
2. In the asset's Basic Information profile, click the **Consumers** count to see a list of all of currently registered consumers for the asset.
3. In the displayed list, click the name of the consumer whose details you want to view or modify.
4. If you want to modify the attributes of the consumer, as displayed in the Basic Information profile, click the **Edit** icon. You can then enter new values for the consumer's fields. Then click the **Save** icon to save the changes.

Unregistering an Existing Consumer

You can now unregister a consumer from an asset in CentraSite.

You might consider unregistering a consumer if you want to:

- Suspend consumption of a particular asset (either on a temporary or permanent basis.)
- Delete an asset permanently from the CentraSite registry.

Note: You can restore the functionality of consumption by re-registering the user or asset with the same asset at any time.

The consequences of unregistering an existing consumer on an asset are as follows:

- The count of consumers in the **Basic Information** profile of the asset is updated.

- Some aspects of runtime such as invocations, performance metrics and other events relating to the operation of a virtual API asset running in Mediator gateway may not work as desired.

Prerequisites for Unregistering:

To unregister an existing consumer from an asset, the following prerequisites must be met:

- The user, or group, or asset should be currently registered as consumer for the specified asset.
- You are the owner of the asset from that you want to unregister the consumer.
- You belong to a role that includes the Modify Assets permission for organization in which the asset resides.
- You have been assigned at least the Modify instance-level permission on the specified asset, or the consumer (which is represented by an individual user, or group, or an asset).

Unregistering an Existing Consumer from the Consumed Asset Details Page

To unregister an existing consumer from the consumed asset details page, proceed as follows:

1. Display the details page of the asset from which you want to unregister the consumer.

The number of assets consuming the displayed asset is typically represented by a numerical prefix to the attribute **Consumers** in the description area of the **Basic Information** profile, for example, **3 Consumers**.
2. Click the Consumers count to see a list of all of currently registered consumers for the asset.
3. In the displayed list, move the mouse over the consumer you want to unregister. This causes a **Delete** icon to appear, that you can use for unregistering.
4. Click the **Delete** icon to unregister the consumer.

A confirmation message appears that the selected consumer will be unregistered from the displayed asset. Click **Yes** to proceed with unregistration.

Note: If you unregister an Application asset, you must manually republish the asset to Mediator gateway to put the changes into effect.

Unregistering an Existing Consumer from the Consumer Asset Details Page

To unregister an existing consumer from the consumer asset details page, proceed as follows:

1. Display the details page of the consumer you want to unregister.

The number of assets consumed by the displayed consumer is typically represented by a numerical prefix to the attribute **Consumed Assets** in the description area of the **Basic Information** profile, for example, **5 Consumed Assets**.

2. Click the **Consumed Assets** count to see a list of all of currently consumed assets.
3. In the displayed list, move the mouse over the asset from which you want to unregister. This causes a **Delete** icon to appear, that you can use for unregistering.
4. Click the **Delete** icon to unregister the consumer.

A confirmation message appears that the displayed consumer will be unregistered from the selected asset. Click **Yes** to proceed with unregistration.

Note: If you unregister an Application asset, you must manually republish the asset to Mediator gateway to put the changes into effect.

4 Run-Time Gateways

■ Introduction to Gateways	60
■ Who Can Create and Manage Gateways?	60
■ Creating and Managing Gateways	61

Introduction to Gateways

To use an instance of CentraSite with webMethods Mediator, API-Portal, or Insight you must define a Mediator gateway, an API-Portal gateway, or an Insight gateway that identifies the specific instance of Mediator, API-Portal, Insight or other policy enforcement point that you want to use. The gateway instance specifies the address of the deployment endpoint, which is the endpoint that CentraSite uses to interact with Mediator or API-Portal to deploy virtual services or virtualized APIs.

Who Can Create and Manage Gateways?

To create and manage gateways, you must have the following roles:

- CentraSite Administrator
- Organization Administrator
- Mediator Administrator role for Mediator gateway
- API-Portal Administrator role for API-Portal gateway
- If you have the CentraSite Administrator role, you can create and manage gateways within any organization.
- If you have the Organization Administrator role, or Mediator, or API-Portal Administrator role for a specific organization, you have the ability to create and manage Mediator , API-Portal, or Insight gateways within that specific organization.

In addition to the above, you can also assign the following permissions to a user for a gateway:

- Instance level full permission
- Instance level view only
- Instance level publish permission

For details about users, groups, roles, and permissions, see *Getting Started with CentraSite*.

Note: A user who belongs to and is assigned permissions to the assets of an organization will not be able to view and publish Mediator gateway assets created for a different organization.

Creating and Managing Gateways

Overview

You can create, update, list, and delete gateways in CentraSite using the following two methods:

- Command Line Interface
- CentraSite Business UI

Managing gateways using CentraSite Business UI is explained in this chapter. Refer to the “Managing Gateways from the Command Line” section in the *CentraSite Administrator’s Guide* for details on the gateway command line tool.

Note: The target model in CentraSite is deprecated starting with the 9.8 release. As a result:

- The **Add Target** action is deprecated in Control UI.
- You cannot edit the target details in Control UI. You can only view details such as: Name, Description, and Target Type of the existing Mediator targets.
- However, you can still deploy, undeploy, and redeploy Virtual APIs to Mediator gateway in Control UI.

Targets created in the previous versions will be migrated as Mediator gateways when you migrate the data. For details see *Upgrading webMethods and Intelligent Business Operations Products*.

Creating a Gateway Asset

Use the following procedure to create a gateway and register it with the CentraSite registry/repository.

Note: This procedure specifies how you can create a Mediator or Insight gateway. For detailed information on creating, listing, updating, deleting an API-Portal gateway, and how to use CentraSite with API-Portal gateway see *Using CentraSite with webMethods API-Portal* section in *Working with the CentraSite Business UI*.

To create a gateway

1. In CentraSite Business UI, click **Manage Governance Rules**.
2. Click the **Add Gateway** action to open the **Create New Gateway** page and complete the following fields:

Note: If you do not see **Add Gateway** in the action bar, it is probably because you are not assigned the required role or do not have the permission to perform the action in CentraSite.

Field	Description
Name	<i>Mandatory.</i> Assign a name to the gateway. The name specified here should be unique for an asset of this type in CentraSite.
Description	<i>Optional.</i> Write a description for the gateway.
Gateway	Select a gateway type. <ul style="list-style-type: none"> ■ API-Portal ■ Mediator ■ Insight
Organization	Choose an organization from the list to which you want to register the Mediator gateway. <p>Note: The drop-down list shows only the organizations to which you are permitted to add Mediator gateways.</p>

3. Configure **CentraSite Communication Information (Mediator to CentraSite)**.

Field	Description
Username	<i>Mandatory.</i> The CentraSite user ID. <p>Note: This user must have permission to update data for the APIs deployed on this Mediator instance, that is the <code>Create Asset</code> or <code>Manage Assets</code> permission. This user should also be part of the <code>MyMediatorGateway Synchronization Group</code> created for this Mediator instance.</p>
Password	<i>Mandatory.</i> The password for the user specified in Username .

Note: The **CentraSite Endpoint** field shows the URL (scheme, host, and port) of the CentraSite Application Server Tier (CAST) in the format, `<scheme>://<host>:<port>`. The scheme is `http` or `https`. The host is the machine

on which CAST is running, and `port` is the port on which CentraSite is listening. The value for the **CentraSite Endpoint** field is determined by the URL that you use to access the CentraSite Business UI.

The CentraSite Communication Information is used to send events, metrics, and other information from Mediator to CentraSite. This information is updated in the CentraSite Communication page under **Mediator Administration** in Integration Server.

4. Configure the **Mediator Communication Information (CentraSite to Mediator)**.

Field	Description
Mediator Endpoint	<i>Mandatory.</i> Specify the URL of the Mediator instance. Format: <code>http://<host>:<port></code> .
Use CentraSite Credentials	Select this option if you want to enable the reuse of existing CentraSite user credentials. If this checkbox is selected, CentraSite automatically disables the subsequent Username and Password fields.
Username	<i>Optional.</i> The Integration Server user who is permitted to deploy assets to this Mediator gateway. By default, only a member of the Integration Server's <code>Administrator</code> group is permitted to deploy assets to this gateway.

Note: This note explains how to permit other users to deploy assets to this target. Mediator exposes several Web service operations to allow CentraSite to manage deployed assets. This Web service is invoked by CentraSite any time a user deploys or undeploys a virtual service or consumer application to Mediator. The **User** and **Password** fields identify an Integration Server user who is permitted to execute the Integration Server services associated with Mediator's deployer service. After installation, only members of the Integration Server's "Administrator" group are permitted to invoke these services. However, administrators have the flexibility to allow their own users or groups to invoke them. Access to these services is controlled by an ACL, called `MediatorDeployer`. Initially, only the predefined "Administrator" group is assigned to this ACL. An Integration Server administrator can remove this group and add other groups or individual users. For example, you can create your own deployer group, (for example, "MyDeployers") and add

Field	Description
	Integration Server user IDs to this group. Then, the user must update the <code>MediatorDeployer</code> ACL by removing the “Administrator” group and adding the “MyDeployers” group. Now, in the User and Password fields on this screen, you can specify any user ID that belongs to the “MyDeployers” group.
Password	<i>Optional.</i> The password of the Integration Server user who is permitted to deployment assets to this gateway. The default password is the Integration Server Administrator password.
Sandbox	<p><i>Optional.</i> Choose the sandbox category to be used to classify the Mediator gateway instance.</p> <ol style="list-style-type: none"> Click Choose. The Sandbox List is displayed with the predefined sandbox categories: Development, Production, and Test. Select a sandbox category from the list. Click OK. For information on the Sandbox categories that CentraSite supports out-of-the-box, in Control UI, go to Administration > Taxonomies. In the Taxonomies page, navigate to Sandbox in the list of taxonomies. If you would like to use sandbox categories that are not supported by CentraSite, you can define custom categories.

- If you have selected *Insight* in the **Gateway** field as a gateway type, configure the following parameters in the **Create Gateway** page:

Field	Description
Name	<i>Mandatory.</i> Assign a name to the gateway. For example, <i>InsightGateway</i> .
Description	Write a description for the gateway.
Gateway	Select <i>Insight</i> as the gateway type from the drop down list.

Field	Description
Organization	Choose an organization from the list to which you want to register the <code>Insight</code> gateway.

6. Click **Publish**.

A gateway instance is created in the specified organization and registered with the CentraSite registry/repository. The Mediator gateway asset details page for the asset that you just created is displayed. For each Mediator gateway that is successfully created, CentraSite creates a *MyMediatorGateway* Synchronization Group, where *MyMediatorGateway* is the name of the Mediator gateway. You can add users who have access to the Mediator gateway to this group so that you can assign permissions to the group instead of assigning permission to individual users.

Configuring Gateway Details

Use the following procedure to configure the Mediator gateway asset details:

1. In CentraSite Business UI, click **Manage Governance Rules**.
2. Search for all the `Mediator` gateway instances. CentraSite lists the gateway instances that you have access to along with a link to access each instance.
3. Click on the instance that you want to configure to open the asset details page.
4. Click **Edit**.
5. If you have logged in as a user who is assigned the `Mediator Administrator` role, you can grant edit permissions to a specified user by clicking on the **Permission** action button.
6. Modify the asset details and click **Save**.

Deleting a Gateway Instance

To delete a Mediator gateway instance:

1. In CentraSite Business UI, click **Manage Governance Rules**.
2. Search for the `Mediator` gateway instance that you want to delete. CentraSite lists only the gateway instances that you have access to along with a link to access each instance.
3. Select the gateway instance that you want to delete.
4. Click **Delete**.

You can also delete the Mediator gateway instance from the asset details page using the **Delete** action.

5 Virtualized Services in CentraSite

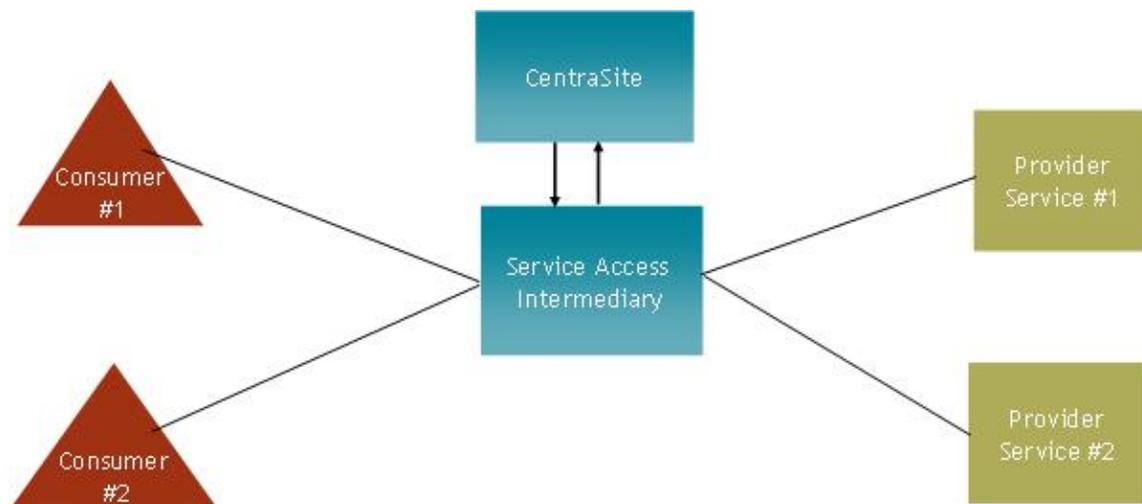
■ Introduction to Virtualized Services	68
■ Who Can Create and Manage Virtual Services?	69
■ Creating Virtual Services	70
■ Configuring SOAP-based Virtual Services	74
■ Viewing REST/XML-based Virtual Services	107
■ Viewing or Editing Virtualized Services	120
■ Creating Run-Time Policies	128
■ Deploying and Undeploying Virtualized Services to Targets	151

Introduction to Virtualized Services

A virtualized service is a service that runs on webMethods Mediator and acts as the consumer-facing proxy for a native service that runs elsewhere on the network. You can have a virtualized service for a SOAP-based web service, a REST service or an XML service. A virtualized service provides a layer of abstraction between the service consumer and the service provider, and promotes loose coupling by providing location, protocol, and format independence between the consuming application and the provider service.

You deploy virtualized services to the webMethods Mediator policy-enforcement point. When a virtualized service is deployed, consumers access this proxy service instead of the actual native service. The virtualized service forwards the request to the appropriate back-end service and, if it has been configured to do so, performs additional mediation functions such as message transformation, protocol bridging, load balancing and failover handling.

The figure below shows webMethods Mediator (the *service access intermediary*) mediating traffic between consumers and virtualized services. Mediator provides loose coupling between consumers and providers, and is able to identify the consumers who are accessing the provider services by examining the traffic flowing through it.



A virtual service is an asset that you create and store in the **Asset Catalog**. You should virtualize each service that you want to deploy to Mediator. You can virtualize a service from scratch or make a virtual copy of an existing service, and then configure the required information in the virtual service definition (VSD) for the service.

You can create one or more virtual services for a single SOAP-based web service. For example, if you have different authentication or routing requirements for two different sets of consumers, you would create a different virtual service for each set of consumers.

The Predefined Asset Types Installed with CentraSite for Virtualization

CentraSite is installed with a predefined set of asset types for virtualization.

The following table identifies the predefined asset types for virtualization and indicates to which native asset type and CentraSite editions they belong.

Native Asset Type Name	Virtual Asset Type Name	Availability in CentraSite Control	Availability in CentraSite Business UI
Web Service	Virtual Service	✓	✓
REST Service	Virtual REST Service		✓
XML Service	Virtual XML Service		✓

This document uses the term *virtualized service* when referring to the three types of virtual services in general.

Important: *This is of specific relevance to virtual REST and XML based service assets. Beginning with version 9.7, CentraSite Control does not support the following out-of-the-box asset types: REST Service, XML Service, Virtual REST Service, and Virtual XML Service. This is because, CentraSite Control does not support the enhanced interface for REST and XML service assets (in contrast, earlier versions of CentraSite Control supported a standardized interface for REST and XML service assets). Documentation of the prior REST and XML service interface is available to Software AG customers who have a current maintenance contract in Empower, Software AG's global extranet (<http://empower.softwareag.com/>).*

Who Can Create and Manage Virtual Services?

To create and manage virtual services, you must belong to a role with the following permissions:

- Create Assets —OR— Manage Assets
- Manage Runtime Policies (required to configure virtual services)
- Mediator Publisher (required to deploy virtual services)

Note: If a user has View permission on a service and Create Assets permission within their own organization, he or she can virtualize the particular web service. However, the user will not be permitted to configure the processing steps for the service that is virtualized unless he or she also has the Manage Runtime Policies permission for their organization. Only users with Manage Runtime Policies permission can configure these steps. Consider identifying a small group of users who will be responsible for configuring the processing steps for a service, and give this group a role that includes the Manage Runtime Policies permission. Because these users might configure virtualized services that other users have created, they will also need Modify permission on the actual virtualized services. To ensure that these users have access to the virtualized services that they need to configure, consider creating a design/change-time policy that automatically gives this group of users Modify permission on a service when it is virtualized.

For more information about roles and permissions, see *Getting Started with CentraSite*.

Creating Virtual Services

Generally speaking, you should ensure that the following conditions are satisfied before you create a virtual service:

- Ensure that the interface for the native service is completely implemented and that interface is reflected in the WSDL or schema file that is registered for the service in the CentraSite registry.
- An instance of the service is deployed and running at a known point in network.
- The metadata for the native service is valid and up-to-date. If the metadata for the native service has not been completely specified or is out-of-date, you should update it before you generate the virtual service so that you do not carry inaccurate/incomplete data into the virtual service.
- *This is of specific relevance to virtual REST and XML based service assets.* Beginning with version 9.7, CentraSite Control does not support the following out-of-the-box asset types: REST Service, XML Service, Virtual REST Service, and Virtual XML Service. This is because, CentraSite Control does not support the enhanced interface for REST and XML service assets (in contrast, earlier versions of CentraSite Control supported a standardized interface for REST and XML service assets). Documentation of the prior REST and XML service interface is available to Software AG customers who have a current maintenance contract in Empower, Software AG's global extranet (<http://empower.softwareag.com/>).

Therefore, you cannot create virtual instances of the asset types REST Service and XML Service using the CentraSite Control user interface. Instead, you can create them by using the CentraSite Business UI.

There are two ways to virtualize a service in CentraSite Control.

Creating a Virtual Service from Scratch

You can create a web service *from scratch*, that is, you manually create the virtual service and set its attributes.

To create a virtual service from scratch

1. In CentraSite Control, go to **Asset Catalog > Browse**.
2. Click **Add Asset**.
3. In the **Add Asset** dialog, specify the following attributes:

In this field...	Do the following...
Type	Choose the Virtual Service asset type.
Name	<p>A name for the new virtual service. Unlike native services, the names of virtual services cannot contain spaces or special characters (except <code>_</code> and <code>-</code>). Consequently, if you adopt a convention that involves using the name of the service as part of the virtual service name, then the names of the services themselves must not contain characters that are invalid in virtual service names.</p> <p>If you want to change the name of a virtual service later, make sure the service is undeployed, and then change the name in the service's detail page.</p>
Description	<i>Optional.</i> A description for the virtual service. This description appears when a user displays a list of virtual services in the user interface.
Organization	Specify the organization to which this virtual service will be added.

Note: The **Organization** list contains the names of all organizations for which you have `Manage Assets` permission.

If you select an organization other than your own organization, you will nevertheless be the owner of the virtual service.

Note: Choose the organization with care. You cannot change the organization assignment after the virtual service is added to the catalog. You can, however, export a virtual service from one organization and import it to another.

- Click **OK**. The virtual service is added to the **Asset Catalog**. The asset's details page is displayed.

Important: During virtualization of an asset, CentraSite will not allow you to add the virtual asset to the asset catalog unless you have specified all "required" attributes in the asset's type definition and all referenced objects to which the asset has an association. The value for the "required" attribute and referenced object must be specified in the virtual service's profile in order to add the asset to the asset catalog.

- View or edit the virtual service's profiles as described in "[Virtualized Services in CentraSite](#)" on page 67.
- Configure the virtual service's processing steps as described in "[Creating Virtual Services](#)" on page 70

Virtualizing an Existing Service

You use the **Virtualize** option to virtualize an existing web service in CentraSite Control.

To virtualize an existing service

- In CentraSite Control, display the service for which you want to create a virtual copy. For procedures, see *CentraSite User's Guide*.
- On the asset detail page, click **Actions** and then select **Virtualize**.

Important: CentraSite will not allow you to virtualize a service if it does not contain an associated WSDL or schema file in the registry.

- In the **Virtualize <name of the type> Service** wizard, specify the following fields:

In this field...	Specify...
Name	A name for the new virtual service. Unlike services, the names of virtual services cannot contain spaces or special characters (except _ and -). Consequently, if you adopt a convention that involves using the name of the service as part of the virtual service name, then the names of the services themselves must not contain characters that are invalid in virtual service names. If you want to change the name of a virtual service later, ensure that the service is undeployed, and then change the name in the service's detail page.
Description	<i>Optional.</i> A description for the virtual service. This description appears when a user displays a list of virtual services in the user interface.

In this field...	Specify...
Organization	<p>Specify the organization to which this virtual service will be added.</p> <p>Note: The Organization list contains the names of all organizations for which you have <code>Manage Assets</code> permission.</p> <p>If you select an organization other than your own organization, you will nevertheless be the owner of the virtual service.</p> <p>Note: Choose the organization with care. You cannot change the organization assignment after the virtual service is added to the catalog. You can, however, export a virtual service from one organization and import it to another.</p>
Version	<p><i>Optional.</i> A version identifier for the virtual service. The version identifier is an optional attribute that you can increment when a service is modified to indicate that the service has been updated. You can use any versioning scheme you choose. The version identifier does not need to be numeric. This is the "public" version identifier that CentraSite Control shows to users when it displays the list of services. Examples:</p> <pre data-bbox="561 1100 1396 1205">0.0a 1.0.0 (beta) Pre-release 001 V1-2007.04.30</pre> <p>You can later create new versions of the virtual service.</p> <p>In addition, CentraSite automatically generates a system version number, which is visible on the virtual service detail page. The system version number is independent from the version number you specify here. For more information, see the <i>CentraSite User's Guide</i>.</p>
Create a Run-Time Policy	<p>Use this checkbox to specify the behavior for the run-time policy (or policies) associated with the virtual service. If you select the checkbox, the run-time policy or policies associated with the virtual service will be created automatically when the service is virtual. If you do not select the checkbox, the run-time policy or policies will not be created when the service is virtual.</p> <p>You can only select the checkbox if you have the <code>Manage Runtime Policies</code> organization-level permission; without this permission, the checkbox is deactivated.</p>

4. If you have attributes specified in the service for virtualizing, click **Next** to go to the attribute mapping dialog.
5. In the **Attributes** dialog, choose the attributes of the service that you want to copy to the virtual service.

Note: Keep the following in mind:

- If you choose a parent attribute, by default *all* its child attributes are copied as well. You can de-select any child attribute.
- Similarly, if you de-select a parent attribute, all its child attributes are de-selected as well.
- Attributes defined as **required** in the asset's type definition and all referenced objects to which the asset has an association will be internally copied from the asset to the virtual service. These attributes will remain selected and disabled by default in the attribute mapping dialog.

6. Click **Finish**.

The virtual service is created in the **Asset Catalog** and its detail page is displayed. The WSDL interface and the entry/exit protocol (for example, HTTP, HTTPS or JMS) will be identical to the service.

7. View or edit the virtual service's profiles as described in "[Creating Virtual Services](#)" on page 70.
8. Configure the virtual service's processing steps as described in "[Creating Virtual Services](#)" on page 70.

Restriction

Even if a Service asset type contains one or more custom profiles, a virtual service of the asset type Service will never include any of these custom profiles in the catalog.

Configuring SOAP-based Virtual Services

The CentraSite Control user interface enables you to configure the following processing steps for a SOAP-based virtual service:

The Entry Protocol Step (SOAP)

The Entry Protocol step specifies the protocol (HTTP, HTTPS or JMS) and SOAP format (1.1 or 1.2) of the requests that the virtual service will accept.

This step allows you to bridge protocols between the consuming application and the native service. For example, suppose you have a native service that is exposed over JMS and a consuming application that submits SOAP requests over HTTP. In this situation,

you can configure the virtual service's Entry Protocol step to accept HTTP requests and configure its Routing Protocols step to route the request to the Web service using JMS.

Besides using the Entry Protocol step to resolve protocol differences between the consumer and the native service, you might use this step to intentionally expose a virtual service over a particular protocol. For example, if you have a native service that is exposed over HTTP, you might expose the virtual service over JMS simply to gain the asynchronous-messaging and guaranteed-delivery benefits that one gains by using JMS as the message transport.

Use the following procedure to configure the Entry Protocol step of a virtual service.

To configure the Entry Protocol step

1. In CentraSite Control, display the details page of SOAP virtual service. If you need procedures for this step, see "[Viewing or Editing Virtualized Services](#)" on page 120.
2. Open the **Processing Steps** profile.
3. Select the **Entry Protocol** tab and specify the protocol (HTTP, HTTPS, or JMS) for the virtual service to accept requests.

Note: CentraSite supports HTTP version 1.1 only.

In this field...	Specify...
Protocol	<p>The protocol (HTTP, HTTPS, or JMS) over which the virtual service will accept requests.</p> <p>Note that you can select <i>both</i> HTTP and HTTPS if needed.</p> <div style="background-color: #f0f0f0; padding: 10px;"> <p>Important Before you deploy a service over HTTPS, ensure that the Integration Server on which the Integration Server is running has been configured for SSL. In addition, make sure you specify an HTTPS port in the Integration Server's <i>Ports Configuration</i> page. (In the Integration Server Administrator, go to Solutions > Mediator > Administration > General and specify the port in the HTTPS Ports Configuration field.) For details on the Port Configuration page, see <i>Administering webMethods Mediator</i>.)</p> </div>
JMS Provider Alias	<p>If you selected JMS, specify the Integration Server's JMS Trigger name. The alias must include the JNDI destination name and the JMS connection factory.</p>
Format	<p>The SOAP format (SOAP 1.1 or SOAP 1.2) of the requests that the virtual service will accept.</p>

The Request Processing Step (SOAP)

The Request Processing step specifies how the SOAP request message is to be transformed or pre-processed before it is submitted to the native service.

As long as a consumer sends a SOAP request to the correct virtual service endpoint, and the request includes a soapAction header, then Mediator can detect the correct service and operation; in this case, no message transformation is required. However, in some cases a virtual service might need to transform SOAP messages.

For example, you might need to accommodate differences between the message content that a consuming application is capable of submitting and the message content that a native service expects. For example, if the consuming application submits an order record using a slightly different structure than the structure expected by the native service, you can use the Request Processing step to transform the record submitted by the consuming application to the structure required by the Web service.

Specifically, you would need to configure the virtual service to:

- Transform or pre-process the request messages into the format required by the native service, before Mediator sends the requests to the native services. Additionally in this case, the transformation is required if the virtual service has a schema validation policy, which validates the requests.
- Transform or pre-process the native service's response messages into the format required by the consumer applications, before Mediator returns the responses to the consumer applications.

There are two ways to transform or pre-process a message:

- By passing the message to an XSLT transformation file.
- By passing the message to a webMethods Integration Server service.

Use the following procedure to configure the Request Processing step of a virtual service.

To configure the Request Processing step

1. In CentraSite Control, display the details page of SOAP virtual service. If you need procedures for this step, see ["Viewing or Editing Virtualized Services" on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Request Processing** tab. On this tab you can specify one or more **Transform** steps and one or more **webMethods IS Service** steps as follows.
 - a. Click **Add Step**, select one of the following kinds of request processing steps and click **OK**.

Request Processing Step	Description
Transform	<p>Configure this step if you want to perform an XSLT message transformation on the request message before it is submitted to the native service.</p> <div data-bbox="771 489 1463 863" style="background-color: #f0f0f0; padding: 10px;"> <p>Important The XSL file uploaded by the user should <i>not</i> contain the XML declaration in it (e.g., <code><?xml version="1.0" encoding="UTF-8"></code>). This is because when the virtual service is deployed to Mediator, Mediator embeds the XSL file in the virtual service definition (VSD), and since the VSD itself is in XML format, there cannot be an XML declaration line in the middle of it. This can lead to unexpected deployment issues which can be avoided by making sure the XSL file does not contain the declaration line.</p> </div>
webMethods IS Service	<p>Configure this step if you want to invoke a webMethods IS service to preprocess the request before it is submitted to the native service. For more information, see "Invoking webMethods IS Services in Virtual Services" on page 261.</p>

- b. In the **Step** list, click the step (**Transform** or **webMethods IS Service**) and complete the appropriate dialog box as follows.

For this step...	Do the following...
Transform	<p>Click Browse and select the XSLT transformation file from your file system and click OK.</p> <div data-bbox="703 1419 1422 1486" style="background-color: #f0f0f0; padding: 10px;"> <p>Note: If you make changes to the XSLT file in the future, you must re-deploy the virtual service.</p> </div>
webMethods IS Service	<p>Type the fully qualified service name or, to display a list of webMethods IS services that are published to CentraSite, type a keyword phrase in the Service field. The wildcard character * is supported. For example, to return all IS services that start with <code>Test</code>, type <code>Test*</code>. (The list that appears also identifies the application server instance on which each service is located.) Then select one or more services to be used to manipulate the request (the axis2 MessageContext instance) and click OK.</p>

For this step...**Do the following...**

Mediator will pass to the invoked IS service the request message context (the axis2 MessageContext instance), which contains the request-specific information. Thus, you can use the public IS services that accept MessageContext as input to manipulate the response contents. For more information, see ["Invoking webMethods IS Services in Virtual Services"](#) on page 261.

Note: The webMethods IS service must be running on the same Integration Server as Mediator.

- c. Configure additional request processing steps if desired, and click **Save**.

Note: Specify the steps in the order in which they should be invoked. Use the arrow buttons to rearrange the sequence of steps. To delete a step, select the check box next to the step and click **Delete**.

The Response Processing Step (SOAP)

The Response Processing step is similar to the Request Processing step. This step specifies how the response message from the native service provider is to be transformed or processed before it is returned to the consuming application.

You may configure and test a virtual service without specifying response processing. You can go back later and specify response processing, if desired.

All steps are optional. You can configure the following steps:

- **Error Messaging:** Configure this step to return a custom error message (and/or the native provider's service fault content) to the consuming application when the native provider returns a service fault. In addition, you can invoke one or more webMethods IS services to process the error message before and/or after the custom error message is added to the response.
- **Transformation:** Configure this step to perform message transformations using a specified XSLT file.
- **webMethods IS Service:** Invoke a user-defined [Using Context Variables in Virtual Services](#) that processes the response

Use the following procedure to configure the Response Processing step of a virtual service.

To configure the Response Processing step

1. In CentraSite Control, display the details page of SOAP virtual service. If you need procedures for this step, see ["Viewing or Editing Virtualized Services"](#) on page 120.
2. Open the **Processing Steps** profile.
3. Select the **Response Processing** tab.
4. Click **Add Step**, select one of the following steps and click **OK**.

You can select only one step at a time, but you can go back and select one or more **Transform** steps, one or more **webMethods IS Service** steps, but only one **Error Messaging** step.

Step	Description
Error Messaging	<p>You use this step to configure error responses for this particular virtual service. Alternatively, you can configure global error responses for <i>all</i> virtual services, using <i>Mediator's Service Fault Configuration</i> page, as described in <i>Administering webMethods Mediator</i>.</p> <p>The precedence of the Error Messaging instructions is as follows:</p> <ul style="list-style-type: none"> ■ If you configure an Error Messaging step for a virtual service, the error messaging step takes precedence over any settings on the global <i>Service Fault Configuration</i> page. ■ If you do not create an Error Messaging step for a virtual service, the settings on the <i>Service Fault Configuration</i> page take precedence.
Transform	<p>Configure this step if you want to invoke an XSLT transformation file to transform the SOAP response payloads from XML format to the format required by the consumer.</p>
webMethods IS Service	<p>Configure this step if you want to invoke a webMethods IS service to process the response message from the native service before it is returned to the consuming application. For more information, see "Invoking webMethods IS Services in Virtual Services" on page 261.</p>

5. If you selected **Error Messaging** above, click **Error Messaging** in the **Step** list and configure it as follows.

You use this step to configure error responses for this particular virtual service. Select one or both of the following options:

Send Native Provider Fault: When you select this option, Mediator returns the native service provider's fault content (if available) to the consuming application. Mediator will send whatever content it received from the native service provider. If you select this option, the **Response** option is ignored when a fault is returned by the native service provider. (Faults returned by internal Mediator exceptions will still be handled by the **Response** option.)

Response: When you select this option, Mediator returns the following fault response to the consuming application:

```
Mediator encountered an error:$ERROR_MESSAGE while executing
operation:$OPERATION service:$SERVICE at time:$TIME on date:$DATE.
The client ip was:$CLIENT_IP. The current user:$USER. The consumer
application:$CONSUMER_APPLICATION".
```

This fault response is returned in both of the following cases:

- When a fault is returned by the native service provider.

In this case, the `$ERROR_MESSAGE` variable in the fault response will contain the message produced by the provider's exception that caused the error. This is equivalent to the `getMessage` call on the Java Exception. This maps to the `faultString` element for SOAP 1.1 or the `Reason` element for SOAP 1.2 catch. Mediator discards the native service provider's fault and does not return this content to the web service caller since it could be considered a security issue, especially if the native provider is returning a stack trace with its response.

- When a fault is returned by internal Mediator exceptions (such as policy violation errors, timeouts, and so on).

In this case, `$ERROR_MESSAGE` will contain the error message generated by Mediator.

The default fault response contains predefined fault handler variables (`$ERROR_MESSAGE`, `$OPERATION`, etc.) which are described in the table below.

You can customize the default fault response using the following substitution variables, where Mediator replaces the variable reference with the real content at run time:

- The predefined context variables listed in [The Predefined Context Variables](#).
- Custom context variables that you declare using Mediator's API (see "[The API for Context Variables](#)" on page 279).

Note: If you want to reference a custom context variable that you have already defined in a context-based routing rule (as opposed to one you have declared using the Mediator API), you must add the prefix `$mx` to the variable name in order to reference the variable. For example, if you defined the variable `TAXID`, you would reference it as `$mx:TAXID`.

The fault handler variables are described below.

Note: If no value is defined for a fault handler variable, then the returned value will be the literal string "null". For example, `$CONSUMER_APPLICATION` will always be "null" if the service's policy does not contain the Identify Consumer action.

Fault Handler Variable	Description
<code>\$ERROR_MESSAGE</code>	The error message produced by the exception that is causing the error. This is equivalent to the <code>getMessage</code> call on the Java Exception. This maps to the <code>faultString</code> element for SOAP 1.1 or the <code>Reason</code> element for SOAP 1.2 catch.
<code>\$OPERATION</code>	The operation that was invoked when this error occurred.
<code>\$SERVICE</code>	The service that was invoked when this error occurred.
<code>\$TIME</code>	The date (as determined on the Container side) at which the error occurred.
<code>\$DATE</code>	The date (as determined on the Container side) at which the error occurred.
<code>\$CLIENT_IP</code>	The IP address of the client invoking the service. This might be available for only certain invoking protocols, such as HTTP(S).
<code>\$USER</code>	The currently authenticated user. The user will be present only if the Transport/SOAP Message have user credentials.
<code>\$CONSUMER_APPLICATION</code>	The currently identified consumer application.

Pre-Processing: Optional. Configure this step if you want to invoke one or more `webMethods IS` services to manipulate the response message before the **Error Messaging** step is invoked. The IS service will have access to the response message context (the `axis2 MessageContext` instance) before it is updated with the custom error message. For example, you might want to send emails or perform custom alerts based on the response payload. For more information, see ["Invoking webMethods IS Services in Virtual Services" on page 261](#).

Post-Processing: Optional. Configure this step if you want to invoke one or more webMethods IS services to manipulate the service fault after the **Error Messaging** step is invoked. The IS service will have access to the entire service fault and the custom error message. You can make further changes to the fault message structure, if needed. For more information, see ["Invoking webMethods IS Services in Virtual Services" on page 261](#).

6. If you selected **Transform** above, click **Transform** in the **Step** list, click **Browse** and select the XSLT transformation file from your file system, then click **OK**. If you make changes to the XSLT file in the future, you must re-deploy the virtual service.
7. If you selected **webMethods IS Service** above, click **webMethods IS Service** in the **Step** list and configure it as follows.

Type the fully qualified service name or, to display a list of webMethods IS services that are published to CentraSite, type a keyword phrase in the **Service** field. The wildcard character * is supported. For example, to return all IS services that start with `Test`, type `Test*`. (The list that appears also identifies the application server instance on which each service is located.) Then select one or more services to be used to manipulate the response (the axis2 MessageContext instance) and click **OK**.

Mediator will pass to the invoked IS service the response message context (the axis2 MessageContext instance), which contains the response-specific information. Thus, you can use the public IS services that accept MessageContext as input to manipulate the response contents. For more information, see ["Invoking webMethods IS Services in Virtual Services" on page 261](#).

Note: The webMethods IS service must be running on the same Integration Server as Mediator.

8. Configure additional response processing steps if desired, and click **Save**.
Arrange the steps in the order in which they should be invoked. Use the arrow buttons to rearrange the sequence of steps. To delete a step, select the check box next to the step and click **Delete**.

The Routing Protocols Step (SOAP)

You can choose to configure one of the following routing protocols for a SOAP-based virtual service.

“Straight Through” Routing (SOAP)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the “Straight Through” routing protocol. (Alternatively, you can choose “Content-based”, “Context-based” or “Load Balancing” routing.)

When you select the “Straight Through” routing protocol, the virtual service will route the requests directly to the native service endpoint you specify. You may specify how to authenticate requests (as with all routing protocols).

Alternatively, you can choose the “Content-Based”, “Context-Based”, or “Load Balancing” routing protocol.

To configure “Straight Through” routing

1. In CentraSite Control, display the details page of SOAP virtual service. If you need procedures for this step, see [“Viewing or Editing Virtualized Services” on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab, specify the following fields, and click **Save**.

Field	Description
HTTP or JMS	Select HTTP .
Routing Type	Select Straight Through
Default To	<p>Click the Endpoint button and select the URL of the native service to route the request to.</p> <p>Alternatively, Mediator offers “Local Optimization” capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the Default To field, specify the native service in either of the following forms:</p> <pre>local://<service_full_path></pre> <p>OR</p> <pre>local://<server>:<port>/ws/<service_full_path></pre> <p>For example:</p> <pre>local://MediatorTestServices:NewMediatorTestServices_Port</pre> <p>which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server. This works for HTTP endpoints only, for all types of Routing Protocols.</p>
Configure Endpoint Properties icon	<p>This icon displays a dialog box that enables you to configure a set of properties for the endpoint as follows:</p> <ul style="list-style-type: none"> ■ SOAP Optimization Method: Optional. Mediator can accept the following optimization methods to optimize the payloads of SOAP requests: <ul style="list-style-type: none"> ■ MTOM: Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism

Field	Description
	<p>(MTOM) attachment, and will forward the attachment to the native service.</p> <ul style="list-style-type: none"> ■ SwA: Indicates that Mediator expects to receive a “SOAP with Attachment” (SwA) request, and will forward the attachment to the native service. ■ None (the default). <ul style="list-style-type: none"> i. Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, Mediator can only forward SwA to the native provider. The same is true for MTOM, and applies to responses received from the native provider. That is, an SwA or MTOM response received by Mediator from a native provider will be forwarded to the caller using the same format it received. ii. When sending SOAP requests that do <i>not</i> contain a MTOM or SWA attachment to a virtual service for a native provider endpoint that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related' (or the virtual service's Request Processing Step should include an Invoking webMethods IS Services in Virtual Services that sets the BUILDER_TYPE context variable to 'multipart/related'). This is necessary so Mediator knows how to parse the response properly. ■ WSS Header Customization: Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native service. <ul style="list-style-type: none"> ■ Pass all security headers: Passes the security header, even if it is processed by Mediator (that is, even if Mediator processes the header according to the virtual service's security run-time policy). <div style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <p>Note: If the virtual service does not contain a security run-time policy, and the <code>mustUnderstand</code> attribute of the security header is <code>0/false</code>, then Mediator will <i>always</i> forward the security header to the native service.</p> </div> <ul style="list-style-type: none"> ■ Remove processed security header from request before routing: Removes the security header if it is processed by Mediator (that is, if Mediator processes the header according to the virtual service's security run-time policy). Note that Mediator will <i>not</i> remove the security header if <i>both</i> of the following conditions are true: 1) Mediator did not process the security

Field	Description
	<p data-bbox="610 310 1382 373">header, and 2) the <code>mustUnderstand</code> attribute of the security header is 0/false).</p> <ul style="list-style-type: none"> <li data-bbox="516 422 1401 653">■ HTTP Connection Timeout: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.connectionTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code> . The default of that property is 30 seconds. <li data-bbox="516 680 1401 911">■ Read Timeout: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.readTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code> . The default of that property is 30 seconds. <li data-bbox="516 959 1401 1094">■ SSL Options: To enable SSL client authentication for the endpoint, you must specify values for both the Client Certificate Alias field and the IS Keystore Alias field. If you specify a value for only one of these fields, a deployment error will occur. <div data-bbox="565 1121 1401 1213" style="background-color: #f0f0f0; padding: 5px;"> <p data-bbox="578 1136 1390 1192">Note: SSL client authentication is optional; you may leave both fields blank.</p> </div> <ul style="list-style-type: none"> <li data-bbox="565 1234 1401 1577">■ Client Certificate Alias: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. <li data-bbox="565 1598 1401 1732">■ IS Keystore Alias: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) will be used for performing SSL client authentication.
<p data-bbox="285 1776 483 1839">HTTP Authentication</p>	<p data-bbox="516 1776 1401 1839">Authentication Scheme: Specify the mode of authentication: Basic Authentication (default), NTLM, OAuth2, or None.</p> <p data-bbox="516 1860 1401 1894">Basic Authentication. Select one of the following options:</p>

Field	Description
	<ul style="list-style-type: none"> ■ Use credentials from incoming request: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the “Authorization” header present in the original client request to the native service. ■ Use specified credentials: Authenticates requests according to the values you specify in the User, Password and Domain fields. <p>NTLM. Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three of the following options for NTLM:</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ Use specified credentials: Mediator uses the values you specify in the User, Password and Domain fields for an NTLM handshake with the server. ■ Transparent: If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>false</code> (the default), then Mediator will behave in “pass by” mode, allowing an NTLM handshake to occur between the client and server. If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>true</code>, then Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication). This property is located in <code>Integration Server_directory \instances \instance_name \config \server.cnf</code>. Note: If the client is a WCF application, then the client should be configured with <code>clientCredentialType</code> set to NTLM. <p>OAuth2. Select one of the following options:</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: Default. This is known as “pass through” mode, in which the consumer includes an OAuth2 access token (a “Bearer” type token) in the request. Mediator then passes the access token unchanged to the native OAuth server. ■ Use specified token: In this mode, the consumer does not include an OAuth2 access token in the request. Instead, the provider generates an OAuth2 access token for each consumer, and Mediator stores the access tokens in Passman. When consumers send requests, Mediator obtains the OAuth2 access tokens from Passman and uses them to access the native services. Specify an

Field	Description
	<p>OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the Show Token button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button.</p> <p>Specify an OAuth access token to be deployed by Mediator by clicking the Show Token button and selecting an OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see "Who Can Create and Manage Virtual Services?" on page 69.</p>
	<p>Note: Keep the following in mind:</p> <ul style="list-style-type: none"> ■ You must set the Integration Server property <code>watt.server.auth.skipForMediator</code> to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (<code>server.cnf</code>), which is located in the <i>Integration Server_directory \config</i> directory. For details, see the <i>webMethods Integration Server Administrator's Guide</i>. ■ The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value HTTP Authentication Token as the identifier) will not be enforced when using the authentication scheme OAuth2.
	<p>None. Select the following option:</p>
	<ul style="list-style-type: none"> ■ Invoke Service Anonymously: Does not authenticate requests.
HTTP Headers	<p>The HTTP headers that you want to use to authenticate the requests.</p> <ul style="list-style-type: none"> ■ Use Existing: Use the HTTP headers that are contained in the requests. ■ Customize: Use the HTTP headers that you specify in the Name and Value columns below. If you need to specify multiple headers, use the plus button to add rows.

"Content-based" Routing (SOAP)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the "Content-based" routing protocol. (Alternatively, you can choose "Straight Through", "Context-based" or "Load Balancing" routing.)

If you have a native service that is hosted at two or more endpoints, you can use the “Content-Based” routing protocol to route specific types of messages to specific endpoints.

You can route messages to different endpoints based on specific values that appear in the request message. You might use this capability, for example, to determine which operation the consuming application has requested, and route requests for complex operations to an endpoint on a fast machine.

The requests are routed according to the content-based routing rules you create. That is, they are routed based on the successful evaluation of one or more XPath expressions that are constructed utilizing the content of the request payload. For example, a routing rule might allow requests for half of the methods of a particular service to be routed to Service A, and the remaining methods to be routed to Service B.

You may also specify how to authenticate requests (as with all routing protocols).

If a SOAP request contains a WS-Security header, Mediator passes it to the native service.

To configure “Content-Based” routing

1. In CentraSite Control, display the details page of SOAP virtual service. If you need procedures for this step, see ["Viewing or Editing Virtualized Services" on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab, specify the following fields and click **Save**.

Field	Description
HTTP or JMS	Select HTTP .
Routing Type	Select Content Based .
Routing Rules	For instructions for creating a routing rule, see "Creating a Routing Rule for “Content-based” Routing (SOAP)" on page 91 .
Default To	<p>A native service endpoint to route the request to in case all routing rules evaluate to False. Click the Endpoint button and select the URL of the native service to route the request to.</p> <p>Alternatively, Mediator offers “Local Optimization” capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the Default To field, specify the native service in either of the following forms:</p>

```
local://<service_full_path>
```

Field	Description
	<p>OR</p> <pre data-bbox="516 380 1247 409">local://<server>:<port>/ws/<service_full_path></pre> <p>For example:</p> <pre data-bbox="516 478 1333 508">local://MediatorTestServices:NewMediatorTestServices_Port</pre> <p>which points to the endpoint service <code>NewMediatorTestServices_Port</code> which is present under the folder <code>MediatorTestServices</code> in Integration Server. This works for HTTP endpoints only, for all types of Routing Protocols.</p>
<p>HTTP Authentication</p>	<p>Authentication Scheme: Specify the mode of authentication: Basic Authentication (default), NTLM, OAuth2, or None.</p> <p>Basic Authentication. Select one of the following options:</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the “Authorization” header present in the original client request to the native service. ■ Use specified credentials: Authenticates requests according to the values you specify in the User, Password and Domain fields. <p>NTLM. Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three of the following options for NTLM:</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ Use specified credentials: Mediator uses the values you specify in the User, Password and Domain fields for an NTLM handshake with the server. ■ Transparent: If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>false</code> (the default), then Mediator will behave in “pass by” mode, allowing an NTLM handshake to occur between the client and server. If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>true</code>, then Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication). This property is located in <code>Integration Server_directory \instances \instance_name \config \server.cnf</code>.

Field	Description
	<p>Note: If the client is a WCF application, then the client should be configured with <code>clientCredentialType</code> set to <code>NTLM</code>.</p>
	<p>OAuth2. Select one of the following options:</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: Default. This is known as “pass through” mode, in which the consumer includes an OAuth2 access token (a “Bearer” type token) in the request. Mediator then passes the access token unchanged to the native OAuth server. ■ Use specified token: In this mode, the consumer does not include an OAuth2 access token in the request. Instead, the provider generates an OAuth2 access token for each consumer, and Mediator stores the access tokens in Passman. When consumers send requests, Mediator obtains the OAuth2 access tokens from Passman and uses them to access the native services. Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the Show Token button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. <p>Specify an OAuth access token to be deployed by Mediator by clicking the Show Token button and selecting an OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see "Who Can Create and Manage Virtual Services?" on page 69.</p>
	<p>Note: Keep the following in mind:</p> <ul style="list-style-type: none"> ■ You must set the Integration Server property <code>watt.server.auth.skipForMediator</code> to “true” and then restart Integration Server for the change to take effect. This property is located in the server configuration file (<code>server.cnf</code>), which is located in the <i>Integration Server_directory \config</i> directory. For details, see the <i>webMethods Integration Server Administrator’s Guide</i>. ■ The run-time actions “Require HTTP Basic Authentication” and “Identify Consumer” (with the value HTTP Authentication Token as the identifier) will not be enforced when using the authentication scheme OAuth2.
	<p>None. Select the following option:</p> <ul style="list-style-type: none"> ■ Invoke Service Anonymously: Does not authenticate requests.

Field	Description
HTTP Headers	<p>The HTTP headers that you want to use to authenticate the requests.</p> <ul style="list-style-type: none"> ■ Use Existing: Use the HTTP headers that are contained in the requests. ■ Customize: Use the HTTP headers that you specify in the Name and Value columns below. If you need to specify multiple headers, use the plus button to add rows.

Creating a Routing Rule for “Content-based” Routing (SOAP)

To create a routing rule

1. Click the **Endpoint** button (next to the **Route To** column).
2. In the **Search for Endpoint** dialog that appears, click the **Search** button to search for the Web service endpoint to route the requests to.
3. Then select a service and click **OK**.
4. Click the **Configure Endpoint Properties** icon (next to the **Endpoint** button) if you want to configure a set of properties for each endpoint individually. In the dialog box, click the endpoint you want to configure and specify the following fields:

Field	Description
SOAP Optimization Method	<p>Optional. Mediator can accept the following optimization methods to optimize the payloads of SOAP requests:</p> <ul style="list-style-type: none"> ■ MTOM: Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment, and will forward the attachment to the native service. ■ SwA: Indicates that Mediator expects to receive a “SOAP with Attachment” (SwA) request, and will forward the attachment to the native service. ■ None (the default). <ul style="list-style-type: none"> a. Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, Mediator can only forward SwA to the native provider. The same is true for MTOM, and applies to responses received from the native provider. That is, an SwA or MTOM response received by Mediator from a native provider will be forwarded to the caller using the same format it received.

Field	Description
	<p>b. When sending SOAP requests that do <i>not</i> contain a MTOM or SWA attachment to a virtual service for a native provider endpoint that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related' (or the virtual service's Request Processing Step should include an Invoking webMethods IS Services in Virtual Services that sets the BUILDER_TYPE context variable to 'multipart/related'). This is necessary so Mediator knows how to parse the response properly.</p>
WSS Header Customization	<p>Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native service.</p> <ul style="list-style-type: none"> ■ Pass all security headers: Passes the security header, even if it is processed by Mediator (that is., even if Mediator processes the header according to the virtual service's security run-time policy). <div style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <p>Note: If the virtual service does not contain a security run-time policy, and the <code>mustUnderstand</code> attribute of the security header is 0/false, then Mediator will <i>always</i> forward the security header to the native service.</p> </div> <ul style="list-style-type: none"> ■ Remove processed security header from request before routing: Removes the security header if it is processed by Mediator (i.e., if Mediator processes the header according to the virtual service's security run-time policy). Note that Mediator will <i>not</i> remove the security header if <i>both</i> of the following conditions are true: 1) Mediator did not process the security header, and 2) the <code>mustUnderstand</code> attribute of the security header is 0/false).
HTTP Connection Timeout	<p>The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.connectionTimeout</code> located in the file <code>Integration Server_directory \packages\WmMediator\config\resources\pg-config.properties</code> . The default of that property is 30 seconds.</p>
Read Timeout	<p>The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.readTimeout</code> located in the file <code>Integration Server_directory \packages\WmMediator\config\resources\pg-config.properties</code> . The default of that property is 30 seconds.</p>

Field	Description
SSL Options	<p>To enable SSL client authentication for the endpoint, you must specify values for both the Client Certificate Alias field and the IS Keystore Alias field. If you specify a value for only one of these fields, a deployment error will occur.</p> <p>Note: SSL client authentication is optional; you may leave both fields blank.</p> <ul style="list-style-type: none"> ■ Client Certificate Alias: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. ■ IS Keystore Alias: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) will be used for performing SSL client authentication.

To create an XPath expression for the routing rule, do the following:

1. Click the **Edit** button (next to the **If True** column).
2. In the XPath Editor that appears, view the **Namespace Map** tab, which displays all predefined namespaces (for example, soapenv, soapenc, and so on.). If you want to add custom namespaces, click **Add Custom Namespace/prefix**, specify a name and value for the namespace and click **OK**. If you need to add additional rows, use the plus button.
3. In the XPath Editor's **All Nodes** tab, expand the namespace's node, choose the method you want for the XPath expression, and click **OK**.
4. In the XPath Editor's **XPATH Evaluator** tab, evaluate the XPath expression by specifying an argument in the **XPath Expression** field, and then click **Evaluate**.
5. After you have evaluated the XPath expression, click **OK**.

"Context-based" Routing (SOAP)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the "Context-based" routing protocol. (Alternatively, you can choose "Straight Through", "Content-based" or "Load Balancing" routing.)

If you have a native service that is hosted at two or more endpoints, you can use the Context-Based protocol to route specific types of messages to specific endpoints.

The requests are routed according to the context-based routing rules you create. A routing rule specifies where requests should be routed, and the criteria by which they should be routed there. For example, requests can be routed according to certain consumers, certain dates/times, or according to requests that exceed/fall below a specified metric (Total Count, Success Count, Fault Count, and so on.). You can create one or more rules.

You may also specify how to authenticate requests (as with all routing protocols).

If a SOAP request contains a WS-Security header, Mediator passes it to the native service.

To configure “Context-Based” Routing

1. In CentraSite Control, display the details page of SOAP virtual service. If you need procedures for this step, see ["Viewing or Editing Virtualized Services" on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab, specify the following fields, and click **Save**.

Field	Description
HTTP or JMS	Select HTTP .
Routing Type	Select Context Based .
Routing Rules	To create a routing rule, click the Add Rule button and complete the Add Routing Rule dialog box. For instructions, see "Creating a Routing Rule for “Context-based” Routing (SOAP)" on page 97 .
Route To	<p>A native service endpoint to route the request to in case all routing rules evaluate to False. Click the Endpoint button and select the URL of the native service to route the request to. To specify additional services, use the plus button next to the field to add rows.</p> <p>Alternatively, Mediator offers “Local Optimization” capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the Default To field, specify the native service in either of the following forms:</p> <pre>local://<service_full_path></pre> <p>OR</p> <pre>local://<server>:<port>/ws/<service_full_path></pre>

Field	Description
	<p>For example:</p> <pre data-bbox="548 373 1398 403">local://MediatorTestServices:NewMediatorTestServices_Port</pre> <p>which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server. This works for HTTP endpoints only, for all types of Routing Protocols.</p>
HTTP Authentication	<p>Authentication Scheme: Specify the mode of authentication: Basic Authentication (default), NTLM, OAuth2 or None.</p> <p>Basic Authentication. Select one of the following options:</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the “Authorization” header present in the original client request to the native service. ■ Use specified credentials: Authenticates requests according to the values you specify in the User, Password and Domain fields. <p>NTLM. Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three of the following options for NTLM:</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ Use specified credentials: Mediator uses the values you specify in the User, Password and Domain fields for an NTLM handshake with the server. ■ Transparent: If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>false</code> (the default), then Mediator will behave in “pass by” mode, allowing an NTLM handshake to occur between the client and server. If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>true</code>, then Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication). This property is located in <code>Integration Server_directory \instances \instance_name \config \server.cnf</code>. Note: If the client is a WCF application, then the client should be configured with <code>clientCredentialType</code> set to NTLM.

Field	Description
	<p>OAuth2. Select one of the following options:</p> <ul style="list-style-type: none"> <li data-bbox="548 380 1398 548">■ Use credentials from incoming request: Default. This is known as “pass through” mode, in which the consumer includes an OAuth2 access token (a “Bearer” type token) in the request. Mediator then passes the access token unchanged to the native OAuth server. <li data-bbox="548 569 1398 978">■ Use specified token: In this mode, the consumer does not include an OAuth2 access token in the request. Instead, the provider generates an OAuth2 access token for each consumer, and Mediator stores the access tokens in Passman. When consumers send requests, Mediator obtains the OAuth2 access tokens from Passman and uses them to access the native services. Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the Show Token button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. <p>Specify an OAuth access token to be deployed by Mediator by clicking the Show Token button and selecting an OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see "Who Can Create and Manage Virtual Services?" on page 69.</p> <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p>Note: Keep the following in mind:</p> <ul style="list-style-type: none"> <li data-bbox="646 1304 1398 1577">■ You must set the Integration Server property <code>watt.server.auth.skipForMediator</code> to “true” and then restart Integration Server for the change to take effect. This property is located in the server configuration file (<code>server.cnf</code>), which is located in the <i>Integration Server_directory</i> \config directory. For details, see the <i>webMethods Integration Server Administrator’s Guide</i>. <li data-bbox="646 1598 1398 1766">■ The run-time actions “Require HTTP Basic Authentication” and “Identify Consumer” (with the value HTTP Authentication Token as the identifier) will not be enforced when using the authentication scheme OAuth2. </div> <p>None. Select the following option:</p> <ul style="list-style-type: none"> <li data-bbox="548 1850 1398 1883">■ Invoke Service Anonymously: Does not authenticate requests.

Field	Description
HTTP Headers	<p>The HTTP headers that you want to use to authenticate the requests.</p> <ul style="list-style-type: none"> ■ Use Existing: Use the HTTP headers that are contained in the requests. ■ Customize: Use the HTTP headers that you specify in the Name and Value columns below. If you need to specify multiple headers, use the plus button to add rows.

Creating a Routing Rule for “Context-based” Routing (SOAP)

To create a routing rule

1. In the **Variable** column, select **Time**, **IP Address**, **Date**, **Consumer**, **Predefined Context Variable** or **Custom Context Variable**. For more information, see ["Using Context Variables in Virtual Services" on page 273](#).
2. In the **Value** column, specify an applicable value. For **Date** choose **Before**, **After** or **Equal To** and enter a date. For **Time** choose **Before** or **After** and enter a time. For **IP Address**, enter numeric values for **Between** and **And**. For **Consumer**, click **Browse** and select a consumer application name. For **Predefined Context Variable** or **Custom Context Variable**, choose the **String** or **Integer** data type. Select a predefined variable name or custom variable name from the drop-down list. For **String**, choose **Equal To** or **Not Equal To** and enter a value. For **Integer**, choose **Greater Than**, **Less Than**, **Not Equal To**, **Equal To** or and enter a value.
 - a. For the list of the predefined context variables, see ["Using Context Variables in Virtual Services" on page 273](#).
 - b. The predefined context variable `PROTOCOL_HEADER` is not available in the drop-down list; to include `PROTOCOL_HEADER` in the rule, define the variable as Custom Context Variable. For more information, see ["The API for Context Variables" on page 279](#).
 - c. If you define a custom context variable in the routing rule, you must write a [Invoking webMethods IS Services in Virtual Services](#) and invoke it in the virtual service's Request Processing step. In this Integration Server service, use the API to get/set the custom context variable. For more information, see ["The API for Context Variables" on page 279](#).
3. If you need to specify multiple variables, use the plus button to add rows.
4. In the **Combination Uses** field, choose an operator for the expression: **AND** (the default) or **OR**.
5. In the **Route To** field, click the **Endpoint** button and choose the URL of the native service to route the request to, if the rule criteria are met.

6. Click the **Configure Endpoint Properties** icon (next to the **Endpoint** button) if you want to configure a set of properties for each endpoint individually. In the dialog box, click the endpoint you want to configure and specify the following fields:

Field	Description
SOAP Optimization Method	<p>Optional. Mediator can accept the following optimization methods to optimize the payloads of SOAP requests:</p> <ul style="list-style-type: none"> ■ MTOM: Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment, and will forward the attachment to the native service. ■ SwA: Indicates that Mediator expects to receive a “SOAP with Attachment” (SwA) request, and will forward the attachment to the native service. ■ None (the default). <p>a. Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, Mediator can only forward SwA to the native provider. The same is true for MTOM, and applies to responses received from the native provider. That is, an SwA or MTOM response received by Mediator from a native provider will be forwarded to the caller using the same format it received.</p> <p>b. When sending SOAP requests that do <i>not</i> contain a MTOM or SWA attachment to a virtual service for a native provider endpoint that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related' (or the virtual service's Request Processing Step should include an Invoking webMethods IS Services in Virtual Services that sets the BUILDER_TYPE context variable to 'multipart/related'). This is necessary so Mediator knows how to parse the response properly.</p>
WSS Header Customization	<p>Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native service.</p> <ul style="list-style-type: none"> ■ Pass all security headers: Passes the security header, even if it is processed by Mediator (that is, even if Mediator processes the header according to the virtual service's security run-time policy).

Note: If the virtual service does not contain a security run-time policy, and the `mustUnderstand`

Field	Description
	<p data-bbox="773 306 1398 405">attribute of the security header is 0/false, then Mediator will <i>always</i> forward the security header to the native service.</p> <ul style="list-style-type: none"> <li data-bbox="643 443 1398 747">■ Remove processed security header from request before routing: Removes the security header if it is processed by Mediator (i.e., if Mediator processes the header according to the virtual service's security run-time policy). Note that Mediator will <i>not</i> remove the security header if <i>both</i> of the following conditions are true: 1) Mediator did not process the security header, and 2) the <code>mustUnderstand</code> attribute of the security header is 0/false).
HTTP Connection Timeout	<p data-bbox="643 795 1398 1035">The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.connectionTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code>. The default of that property is 30 seconds.</p>
Read Timeout	<p data-bbox="643 1079 1398 1318">The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.readTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code>. The default of that property is 30 seconds.</p>
SSL Options	<p data-bbox="643 1362 1398 1497">To enable SSL client authentication for the endpoint, you must specify values for both the Client Certificate Alias field and the IS Keystore Alias field. If you specify a value for only one of these fields, a deployment error will occur.</p> <div data-bbox="643 1514 1398 1612" style="background-color: #f0f0f0; padding: 5px;"> <p data-bbox="656 1530 1398 1593">Note: SSL client authentication is optional; you may leave both fields blank.</p> </div> <ul style="list-style-type: none"> <li data-bbox="643 1629 1398 1902">■ Client Certificate Alias: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests

Field	Description
	<p>or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses.</p> <ul style="list-style-type: none"> ■ IS Keystore Alias: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) will be used for performing SSL client authentication.

7. Click **OK**.

“Load Balancing” Routing (SOAP)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the “Load Balancing” routing protocol. (Alternatively, you can choose “Straight Through”, “Content-Based” or “Context-Based” routing.)

If you have a Web service that is hosted at two or more endpoints, you can use the Load Balancing option to distribute requests among the endpoints.

Requests are distributed across multiple endpoints. The requests are intelligently routed based on the “round-robin” execution strategy. The load for a service is balanced by directing requests to two or more services in a pool, until the optimum level is achieved. The application routes requests to services in the pool sequentially, starting from the first to the last service without considering the individual performance of the services. After the requests have been forwarded to all the services in the pool, the first service is chosen for the next loop of forwarding.

Load-balanced endpoints also have automatic Failover capability. If a load-balanced endpoint is unavailable (for example, if a connection is refused), then that endpoint is marked as “down” for the number of seconds you specify in the “Suspend the Failed Endpoint” field (during which the endpoint will not be used for sending the request), and the next configured endpoint is tried. If all the configured load-balanced endpoints are down, then a SOAP fault is sent back to the client. After the suspension period expires, each endpoint marked will be available again to send the request.

To configure “Load Balancing” routing

1. In CentraSite Control, display the details page of SOAP virtual service. If you need procedures for this step, see [“Viewing or Editing Virtualized Services” on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab, specify the following fields, and click **Save**.

In this field...	Specify...
HTTP or JMS	HTTP.

In this field...	Specify...
Routing Type	Load Balancing
Route To	<p>The URLs of two or more native services in a pool to which the requests will be routed. The application routes the requests to services in the pool sequentially, starting from the first to the last service without considering the individual performance of the services. After the requests have been forwarded to all the services in the pool, the first service is chosen for the next loop of forwarding.</p> <p>To specify the first service, click the Endpoint button and select the URL of the Web service to route the request to. To specify additional services, use the plus button next to the field to add rows.</p> <p>Alternatively, Mediator offers “Local Optimization” capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the Default To field, specify the native service in either of the following forms:</p> <pre>local://<service_full_path></pre> <p>OR</p> <pre>local://<server>:<port>/ws/<service_full_path></pre> <p>For example:</p> <pre>local://MediatorTestServices:NewMediatorTestServices_Port</pre> <p>which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server. This works for HTTP endpoints only, for all types of Routing Protocols.</p>
Configure Endpoint Properties icon	<p>This button displays a dialog box that enables you to configure a single set of properties that will be shared by all the endpoints. In the dialog box, specify the following fields:</p> <ul style="list-style-type: none"> ■ SOAP Optimization Method: Optional. Mediator can accept the following optimization methods to optimize the payloads of SOAP requests: <ul style="list-style-type: none"> ■ MTOM: Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment, and will forward the attachment to the native service.

In this field...	Specify...
	<ul style="list-style-type: none"> ■ SwA: Indicates that Mediator expects to receive a “SOAP with Attachment” (SwA) request, and will forward the attachment to the native service. ■ None (the default). <ol style="list-style-type: none"> 1. Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, Mediator can only forward SwA to the native provider. The same is true for MTOM, and applies to responses received from the native provider. That is, an SwA or MTOM response received by Mediator from a native provider will be forwarded to the caller using the same format it received. 2. When sending SOAP requests that do <i>not</i> contain a MTOM or SWA attachment to a virtual service for a native provider endpoint that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related' (or the virtual service's Request Processing Step should include an Invoking webMethods IS Services in Virtual Services that sets the BUILDER_TYPE context variable to 'multipart/related'). This is necessary so Mediator knows how to parse the response properly. <ul style="list-style-type: none"> ■ WSS Header Customization: Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native service. <ul style="list-style-type: none"> ■ Pass all security headers: Passes the security header, even if it is processed by Mediator (that is, even if Mediator processes the header according to the virtual service's security run-time policy). <div data-bbox="609 1381 1414 1556" style="background-color: #f0f0f0; padding: 5px; margin: 5px 0;"> <p>Note: If the virtual service does not contain a security run-time policy, and the <code>mustUnderstand</code> attribute of the security header is <code>0/false</code>, then Mediator will <i>always</i> forward the security header to the native service.</p> </div> ■ Remove processed security header from request before routing: Removes the security header if it is processed by Mediator (that is, if Mediator processes the header according to the virtual service's security run-time policy). Note that Mediator will <i>not</i> remove the security header if <i>both</i> of the following conditions are true: 1) Mediator did not process the security header, and 2) the <code>mustUnderstand</code> attribute of the security header is <code>0/false</code>).

In this field...	Specify...
	<ul style="list-style-type: none"> <li data-bbox="516 323 1425 562">■ HTTP Connection Timeout: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.connectionTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code> . The default of that property is 30 seconds. <li data-bbox="516 583 1425 823">■ Read Timeout: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.readTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code> . The default of that property is 30 seconds. <li data-bbox="516 865 1425 1008">■ SSL Options: To enable SSL client authentication for the endpoint, you must specify values for both the Client Certificate Alias field and the IS Keystore Alias field. If you specify a value for only one of these fields, a deployment error will occur.
	<p>Note: SSL client authentication is optional; you may leave both fields blank.</p>
	<ul style="list-style-type: none"> <li data-bbox="565 1138 1425 1486">■ Client Certificate Alias: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. <li data-bbox="565 1507 1425 1638">■ IS Keystore Alias: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) will be used for performing SSL client authentication.
<p>Suspend the Failed Endpoint</p>	<p>A numeric timeout value (in seconds).</p> <p>Default: 30. When this timeout value expires, the system routes the execution of the virtual service to the next consecutive Web service endpoint specified in the Route To field.</p>

In this field...	Specify...
HTTP Authentication	<p data-bbox="516 327 1401 390">Authentication Scheme: Specify the mode of authentication: Basic Authentication (default), NTLM, OAuth2 or None.</p> <p data-bbox="516 415 1401 445">Basic Authentication. Select one of the following options:</p> <ul data-bbox="516 470 1401 688" style="list-style-type: none"> <li data-bbox="516 470 1401 604">■ Use credentials from incoming request: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the “Authorization” header present in the original client request to the native service. <li data-bbox="516 625 1401 688">■ Use specified credentials: Authenticates requests according to the values you specify in the User, Password and Domain fields. <p data-bbox="516 735 1401 974">NTLM. Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three of the following options for NTLM:</p> <ul data-bbox="516 999 1401 1579" style="list-style-type: none"> <li data-bbox="516 999 1401 1092">■ Use credentials from incoming request: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. <li data-bbox="516 1117 1401 1209">■ Use specified credentials: Mediator uses the values you specify in the User, Password and Domain fields for an NTLM handshake with the server. <li data-bbox="516 1234 1401 1579">■ Transparent: If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>false</code> (the default), then Mediator will behave in “pass by” mode, allowing an NTLM handshake to occur between the client and server. If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>true</code>, then Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication). This property is located in <code>Integration Server_directory \instances \instance_name \config \server.cnf</code>. Note: If the client is a WCF application, then the client should be configured with <code>clientCredentialType</code> set to <code>NTLM</code>. <p data-bbox="516 1625 1401 1654">OAuth2. Select one of the following options:</p> <ul data-bbox="516 1680 1401 1898" style="list-style-type: none"> <li data-bbox="516 1680 1401 1814">■ Use credentials from incoming request: Default. This is known as “pass through” mode, in which the consumer includes an OAuth2 access token (a “Bearer” type token) in the request. Mediator then passes the access token unchanged to the native OAuth server. <li data-bbox="516 1839 1401 1898">■ Use specified token: In this mode, the consumer does not include an OAuth2 access token in the request. Instead, the provider

In this field...	Specify...
	<p>generates an OAuth2 access token for each consumer, and Mediator stores the access tokens in Passman. When consumers send requests, Mediator obtains the OAuth2 access tokens from Passman and uses them to access the native services. Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the Show Token button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button.</p> <p>Specify an OAuth access token to be deployed by Mediator by clicking the Show Token button and selecting an OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see "Who Can Create and Manage Virtual Services?" on page 69.</p> <p>Note: Keep the following in mind:</p> <ul style="list-style-type: none"> ■ You must set the Integration Server property <code>watt.server.auth.skipForMediator</code> to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (<code>server.cnf</code>), which is located in the <i>Integration Server_directory \config</i> directory. For details, see the <i>webMethods Integration Server Administrator's Guide</i>. ■ The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value HTTP Authentication Token as the identifier) will not be enforced when using the authentication scheme OAuth2. <p>None. Select the following option:</p> <ul style="list-style-type: none"> ■ Invoke Service Anonymously: Does not authenticate requests.
HTTP Headers	<p>The HTTP headers that you want to use to authenticate the requests.</p> <ul style="list-style-type: none"> ■ Use Existing: Use the HTTP headers that are contained in the requests. ■ Customize: Use the HTTP headers that you specify in the Name and Value columns below. If you need to specify multiple headers, use the plus button to add rows.

The Routing Protocol for Services Exposed over JMS (SOAP)

You can use the Routing Protocols step to specify a JMS queue to which the Mediator is to submit the request, and the destination to which the native service is to return the response.

To configure the Routing Protocols step for virtual services exposed over JMS

1. In CentraSite Control, display the details page of SOAP virtual service. For procedures for this step, see ["Viewing or Editing Virtualized Services" on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab and select the **JMS** button. On this tab specify the following JMS routing protocol fields, and click **Save**.

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

In this field...	Specify...
JMS URI	<p>A connection alias for connecting to the JMS provider (for example, an Integration Server alias or a JNDI URL). For example, a JNDI URL of the form:</p> <pre>jms:queue:dynamicQueues/MyRequestQueue? wm-wsendpointalias=MediatorConsumer &targetService=vs-jms-in-echo</pre> <p>Note that the <code>wm-wsendpointalias</code> parameter is required for Integration Server/Mediator to look up the JMS consumer alias to send the request to the specified queue (for example, <code>MyRequestQueue</code>), which is a dynamic queue in ActiveMQ. Also, the <code>targetService</code> parameter is required if sending to another virtual service that uses JMS as the entry protocol.</p>
Priority	Optional. A numeric value that specifies the priority of the JMS message in the queue.
Reply to Destination	Optional. A queue name for the incoming JMS request.
Time to Live	Optional. A numeric value (in milliseconds) that specifies the lifespan of the JMS message.
Delivery Mode	Optional. The type of message delivery to the endpoint. <ul style="list-style-type: none"> ■ None (default).

In this field...	Specify...
	<ul style="list-style-type: none"> ■ Persistent: The message is stored by the JMS server before delivering it to the consumer. ■ Non-Persistent: The message is not stored before delivery.
Message Properties	<p>The message properties to use.</p> <ul style="list-style-type: none"> ■ Use Existing (default): Use existing properties. ■ Customize: Specify one or more property names and values. To add additional rows, use the plus button.
JMS Headers	<p>The JMS headers that you want to use to authenticate the requests.</p> <ul style="list-style-type: none"> ■ Use Existing (default): Use existing headers. ■ Customize: Specify one or more header names and values. To add additional rows, use the plus button.

Viewing REST/XML-based Virtual Services

The CentraSite Control user interface enables you to view and examine the processing steps for a virtual REST or XML service.

Important: *This is of specific relevance to REST and XML based virtual service assets.* Beginning with version 9.7, CentraSite supports the enhanced interface for REST and XML service assets (in contrast, earlier versions of CentraSite supported a standardized interface for REST and XML service assets). Note that the standardized REST and XML service interface that was implemented by versions of CentraSite prior to version 9.7 is not compatible with the enhanced interface that is implemented by current version of CentraSite. Documentation of the prior REST and XML service interface is available to Software AG customers who have a current maintenance contract in Empower, Software AG's global extranet (<http://empower.softwareag.com/>).

If You Migrate REST/XML Services from a Pre-9.7 Release

If you have REST and XML services that were created prior to version 9.7, these REST services will continue to hold the old version's metadata in the enhanced REST service interface implemented by current version of CentraSite. Keep in mind that you can only view properties of the REST and XML services in CentraSite Control. You cannot configure properties of the REST or XML services using the CentraSite Control user interface (not even if you belong to the CentraSite Administrator role). You will only be able to configure the properties using the CentraSite Business UI.

The Entry Protocol Step (REST/XML)

Perform the following steps to view and examine the **Entry Protocol** step for a virtual REST or XML service.

To examine the Entry Protocol step

1. In CentraSite Control, display the details page of REST/XML virtual service. For procedures for this step, see ["Viewing or Editing Virtualized Services" on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Entry Protocol** tab. On this tab, examine the following fields:

Field	Describes
Protocol	Specifies the protocol (HTTP and/or HTTPS) over which the virtual REST service or virtual XML service accepts requests.
HTTP Methods	Specifies the HTTP methods (GET, POST, PUT, DELETE) that the virtual service should be allowed to perform on a REST resource.

The Request Processing Step (REST/XML)

Perform the following steps to view and examine the **Request Processing** step for a virtual REST or XML service.

To examine the Request Processing step

1. In CentraSite Control, display the details page of REST/XML virtual service. For procedures for this step, see ["Viewing or Editing Virtualized Services" on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Request Processing** tab. On this tab, click the step **Transform** or **webMethods IS Service**, and examine the following fields:

Step	Description
Transform	Specifies the XSLT transformation file used to perform an XSLT message transformation on the request message before it is submitted to the native REST/XML service.
webMethods IS Service	Specifies the webMethods IS services that is used to manipulate the request (the axis2 MessageContext instance).

The Response Processing Step (REST/XML)

Perform the following steps to view and examine the **Response Processing** step for a virtual REST or XML service.

To examine the Response Processing step

1. In CentraSite Control, display the details page of REST/XML virtual service. For procedures for this step, see ["Viewing or Editing Virtualized Services"](#) on page 120.
2. Open the **Processing Steps** profile.
3. Select the **Response Processing** tab. This tab includes one **Error Messaging** step, one or more **Transform** steps and one or more **webMethods IS Service** steps as follows.

Step	Description
Error Messaging	Specifies the error responses for the virtual REST/XML service. Alternatively, you can have global error responses for all virtual services that are configured using Mediator's Service Fault Configuration screen, as described in <i>Administering webMethods Mediator</i> .
Transform	Specifies the XSLT transformation file to transform the response payloads from XML format to the format required by the consumer.
webMethods IS Service	Specifies the webMethods IS services to process the response message from the native service before it is returned to the consuming application. For more information, see "Invoking webMethods IS Services in Virtual Services" on page 261.

The Routing Protocols Step (REST/XML)

If your Entry Protocol is HTTP or HTTPS, you can have one of the following routing protocols for a virtual REST/ XML service.

The "Straight Through" Routing Protocol Step (REST/XML)

Perform the following steps to examine the "Straight Through" routing of a virtual REST or XML service.

To examine “Straight Through” routing

1. In CentraSite Control, display the details page of REST/XML virtual service. For procedures for this step, see ["Viewing or Editing Virtualized Services"](#) on page 120.
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab. On this tab, examine the following fields.

Field	Description
Service Type	Specifies XML (for a native REST or XML service).
Routing Type	Specifies the routing type Straight Through .
HTTP Method	Specifies the HTTP method to pass to the native service.
Default To	Specifies the URL of the native service to route the request to.
Configure Endpoint Properties icon	<p>This icon displays a dialog box that enables you to examine the properties defined for an endpoint as follows:</p> <ul style="list-style-type: none"> ■ HTTP Connection Timeout: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.connectionTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code>. The default of that property is 30 seconds. ■ Read Timeout: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.readTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code>. The default of that property is 30 seconds. ■ SSL Options: Enables SSL client authentication for the endpoint. <ul style="list-style-type: none"> ■ Client Certificate Alias: The client's private key used for performing SSL client authentication. ■ IS Keystore Alias: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication.

Field	Description
HTTP Authentication	<p data-bbox="505 327 1336 390">Authentication Scheme: Specifies the mode of authentication: Basic Authentication (default), NTLM, OAuth2 or None.</p> <p data-bbox="505 436 748 466">Basic Authentication.</p> <ul data-bbox="505 491 1425 716" style="list-style-type: none"> <li data-bbox="505 491 1425 625">■ Use credentials from incoming request: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the “Authorization” header present in the original client request to the native service. <li data-bbox="505 646 1425 716">■ Use specified credentials: Authenticates requests according to the values you specify in the User, Password, and Domain fields. <p data-bbox="505 758 1425 995">NTLM. Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three of the following options for NTLM:</p> <ul data-bbox="505 1020 1425 1604" style="list-style-type: none"> <li data-bbox="505 1020 1425 1119">■ Use credentials from incoming request: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. <li data-bbox="505 1140 1425 1239">■ Use specified credentials: Mediator uses the values specifies in the User, Password, and Domain fields for an NTLM handshake with the server. <li data-bbox="505 1260 1425 1604">■ Transparent: If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>false</code> (the default), then Mediator will behave in “pass by” mode, allowing an NTLM handshake to occur between the client and server. If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>true</code>, then Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication). This property is located in <code>Integration Server_directory \instances \instance_name \config \server.cnf</code>. Note: If the client is a WCF application, then the client should be configured with <code>clientCredentialType</code> set to NTLM. <p data-bbox="505 1646 597 1675">OAuth2.</p> <ul data-bbox="505 1701 1425 1923" style="list-style-type: none"> <li data-bbox="505 1701 1425 1835">■ Use credentials from incoming request: Default. This is known as “pass through” mode, in which the consumer includes an OAuth2 access token (a “Bearer” type token) in the request. Mediator then passes the access token unchanged to the native OAuth server. <li data-bbox="505 1856 1425 1923">■ Use specified token: In this mode, the consumer does not include an OAuth2 access token in the request. Instead, the provider

Field	Description
	<p>generates an OAuth2 access token for each consumer, and Mediator stores the access tokens in Passman. When consumers send requests, Mediator obtains the OAuth2 access tokens from Passman and uses them to access the native services. Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the Show Token button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button.</p> <p>None. Select the following option:</p> <ul style="list-style-type: none"> ■ Invoke Service Anonymously: Does not authenticate requests.
HTTP Headers	<p>Specifies the HTTP headers used to authenticate the requests.</p> <ul style="list-style-type: none"> ■ Use Existing: Uses the HTTP headers that are contained in the requests. ■ Customize: Uses the HTTP headers that are specified in the Name and Value columns below.

The “Content-based” Routing Protocol Step (REST or XML)

Perform the following steps to examine the “Content-Based” routing of a virtual REST or XML service.

To examine “Content-Based” routing

1. In CentraSite Control, display the details page of REST/XML virtual service. For procedures for this step, see [“Viewing or Editing Virtualized Services” on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab. On this tab, examine the following fields:

Field	Description
Service Type	Specifies XML (for a native REST or XML service).
Routing Type	Specifies the routing type Content-Based .
Routing Rules	<p>Specifies one or more routing rules.</p> <p>Click the Configure Endpoint Properties icon (next to the Endpoint button) for an endpoint.</p>

Field	Description
	<p>This icon displays a dialog box that enables you to examine the properties defined for an endpoint as follows:</p> <ul style="list-style-type: none"> <li data-bbox="516 436 1401 674">■ HTTP Connection Timeout: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.connectionTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code>. The default of that property is 30 seconds. <li data-bbox="516 699 1401 936">■ Read Timeout: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.readTimeout</code> located in the file <code>Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties</code>. The default of that property is 30 seconds. <li data-bbox="516 961 1401 1234">■ SSL Options: Enables SSL client authentication for the endpoint. <ul style="list-style-type: none"> <li data-bbox="565 1010 1401 1077">■ Client Certificate Alias: The client's private key used for performing SSL client authentication. <li data-bbox="565 1102 1401 1234">■ IS Keystore Alias: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication.
HTTP Method	Specifies the HTTP method to pass to the native service.
Default To	Specifies the URL of the native service to route the request to.
HTTP Authentication	<p>Authentication Scheme: Specifies the mode of authentication: Basic Authentication (default), NTLM, OAuth2 or None.</p> <p>Basic Authentication.</p> <ul style="list-style-type: none"> <li data-bbox="516 1598 1401 1730">■ Use credentials from incoming request: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. <li data-bbox="516 1755 1401 1816">■ Use specified credentials: Authenticates requests according to the values you specify in the User, Password, and Domain fields.

Field	Description
	<p>NTLM. Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three of the following options for NTLM:</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ Use specified credentials: Mediator uses the values specifies in the User, Password, and Domain fields for an NTLM handshake with the server. ■ Transparent: If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>false</code> (the default), then Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>true</code>, then Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication). This property is located in <code>Integration Server_directory \instances \instance_name \config \server.cnf</code>. Note: If the client is a WCF application, then the client should be configured with <code>clientCredentialType</code> set to NTLM.
	<p>OAuth2.</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: Default. This is known as “pass through” mode, in which the consumer includes an OAuth2 access token (a “Bearer” type token) in the request. Mediator then passes the access token unchanged to the native OAuth server. ■ Use specified token: In this mode, the consumer does not include an OAuth2 access token in the request. Instead, the provider generates an OAuth2 access token for each consumer, and Mediator stores the access tokens in Passman. When consumers send requests, Mediator obtains the OAuth2 access tokens from Passman and uses them to access the native services. Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the Show Token button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. <p>None. Select the following option:</p>

Field	Description
	<ul style="list-style-type: none"> ■ Invoke Service Anonymously: Does not authenticate requests.
HTTP Headers	<p>Specifies the HTTP headers used to authenticate the requests.</p> <ul style="list-style-type: none"> ■ Use Existing: Uses the HTTP headers that are contained in the requests. ■ Customize: Uses the HTTP headers that are specified in the Name and Value columns below.

The “Context-based” Routing Protocol Step (REST or XML)

Perform the following steps to examine the “Context-Based” routing of a virtual REST or XML service.

To examine “Context-Based” Routing

1. In CentraSite Control, display the details page of REST/XML virtual service. For procedures for this step, see ["Viewing or Editing Virtualized Services" on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab. On this tab, examine the following fields.

Field	Description
Service Type	Specifies XML (for a native REST or XML service).
Routing Type	Specifies the routing type Context-Based .
Routing Rules	<p>Specifies one or more routing rules.</p> <p>Click the Configure Endpoint Properties icon (next to the Endpoint button) for an endpoint.</p>

This icon displays a dialog box that enables you to examine the properties defined for an endpoint as follows:

- **HTTP Connection Timeout:** The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file `Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties`. The default of that property is 30 seconds.

Field	Description
	<ul style="list-style-type: none"> <li data-bbox="505 323 1409 533">■ Read Timeout: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property <code>pg.endpoint.readTimeout</code> located in the file <i>Integration Server_directory\packages\WmMediator\config\resources\pg-config.properties</i>. The default of that property is 30 seconds. <li data-bbox="505 554 1409 827">■ SSL Options: Enables SSL client authentication for the endpoint. <ul style="list-style-type: none"> <li data-bbox="553 606 1409 674">■ Client Certificate Alias: The client's private key used for performing SSL client authentication. <li data-bbox="553 695 1409 827">■ IS Keystore Alias: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication.
HTTP Method	Specifies the HTTP method to pass to the native service.
Default To	Specifies the URL of the native service to route the request to.
HTTP Authentication	<p data-bbox="505 1024 1333 1087">Authentication Scheme: Specifies the mode of authentication: Basic Authentication (default), NTLM, OAuth2 or None.</p> <p data-bbox="505 1140 748 1165">Basic Authentication.</p> <ul style="list-style-type: none"> <li data-bbox="505 1192 1409 1325">■ Use credentials from incoming request: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the “Authorization” header present in the original client request to the native service. <li data-bbox="505 1346 1409 1413">■ Use specified credentials: Authenticates requests according to the values you specify in the User, Password, and Domain fields. <p data-bbox="505 1461 1409 1692">NTLM. Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three of the following options for NTLM:</p> <ul style="list-style-type: none"> <li data-bbox="505 1719 1409 1812">■ Use credentials from incoming request: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server.

Field	Description
	<ul style="list-style-type: none"> <li data-bbox="505 323 1408 428">■ Use specified credentials: Mediator uses the values specified in the User, Password, and Domain fields for an NTLM handshake with the server. <li data-bbox="505 449 1408 789">■ Transparent: If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>false</code> (the default), then Mediator will behave in “pass by” mode, allowing an NTLM handshake to occur between the client and server. If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>true</code>, then Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication). This property is located in <code>Integration Server_directory \instances \instance_name \config \server.cnf</code>. Note: If the client is a WCF application, then the client should be configured with <code>clientCredentialType</code> set to NTLM.
	<p data-bbox="505 831 597 861">OAuth2.</p> <ul style="list-style-type: none"> <li data-bbox="505 882 1408 1020">■ Use credentials from incoming request: Default. This is known as “pass through” mode, in which the consumer includes an OAuth2 access token (a “Bearer” type token) in the request. Mediator then passes the access token unchanged to the native OAuth server. <li data-bbox="505 1041 1408 1419">■ Use specified token: In this mode, the consumer does not include an OAuth2 access token in the request. Instead, the provider generates an OAuth2 access token for each consumer, and Mediator stores the access tokens in Passman. When consumers send requests, Mediator obtains the OAuth2 access tokens from Passman and uses them to access the native services. Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the Show Token button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. <p data-bbox="505 1440 935 1470">None. Select the following option:</p> <ul style="list-style-type: none"> <li data-bbox="505 1491 1305 1520">■ Invoke Service Anonymously: Does not authenticate requests.
HTTP Headers	<p data-bbox="505 1566 1295 1596">Specifies the HTTP headers used to authenticate the requests.</p> <ul style="list-style-type: none"> <li data-bbox="505 1617 1338 1688">■ Use Existing: Uses the HTTP headers that are contained in the requests. <li data-bbox="505 1709 1377 1772">■ Customize: Uses the HTTP headers that are specified in the Name and Value columns below.

The “Load Balancing” Routing Protocol Step (REST or XML)

Perform the following steps to examine the “Load Balancing” routing of a virtual REST or XML service.

To examine “Load Balancing” routing

1. In CentraSite Control, display the details page of REST/XML virtual service. For procedures for this step, see [“Viewing or Editing Virtualized Services” on page 120](#).
2. Open the **Processing Steps** profile.
3. Select the **Routing Protocols** tab. On this tab, examine the following fields.

Field	Description
Service Type	Specifies XML (for a native REST or XML service).
Routing Type	Specifies the routing type Load Balancing .
Routing Rules	Specifies one or more routing rules. Click the Configure Endpoint Properties icon (next to the Endpoint button) for an endpoint.

This icon displays a dialog box that enables you to examine the properties defined for an endpoint as follows:

- **HTTP Connection Timeout:** The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file `Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties`. The default of that property is 30 seconds.
- **Read Timeout:** The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file `Integration Server_directory \packages \WmMediator \config \resources \pg-config.properties`. The default of that property is 30 seconds.
- **SSL Options:** Enables SSL client authentication for the endpoint.
 - **Client Certificate Alias:** The client's private key used for performing SSL client authentication.
 - **IS Keystore Alias:** The keystore alias of the instance of Integration Server on which Mediator is running. This value

Field	Description
	(along with the value of Client Certificate Alias) is used for performing SSL client authentication.
HTTP Method	Specifies the HTTP method to pass to the native service.
Default To	Specifies the URL of the native service to route the request to.
HTTP Authentication	Authentication Scheme: Specifies the mode of authentication: Basic Authentication (default), NTLM , OAuth2 or None .
	<p data-bbox="505 678 748 709">Basic Authentication.</p> <ul data-bbox="505 730 1422 961" style="list-style-type: none"> <li data-bbox="505 730 1422 877">■ Use credentials from incoming request: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the “Authorization” header present in the original client request to the native service. <li data-bbox="505 888 1422 961">■ Use specified credentials: Authenticates requests according to the values you specify in the User, Password, and Domain fields.
	<p data-bbox="505 1003 1422 1245">NTLM. Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three of the following options for NTLM:</p> <ul data-bbox="505 1266 1422 1848" style="list-style-type: none"> <li data-bbox="505 1266 1422 1360">■ Use credentials from incoming request: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. <li data-bbox="505 1381 1422 1476">■ Use specified credentials: Mediator uses the values specifies in the User, Password, and Domain fields for an NTLM handshake with the server. <li data-bbox="505 1497 1422 1848">■ Transparent: If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>false</code> (the default), then Mediator will behave in “pass by” mode, allowing an NTLM handshake to occur between the client and server. If the property <code>watt.pg.disableNtlmAuthHandler</code> is set to <code>true</code>, then Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication). This property is located in <code>Integration Server_directory \instances \instance_name \config \server.cnf</code>. Note: If the client is a WCF application, then the client should be configured with <code>clientCredentialType</code> set to <code>NTLM</code>.

Field	Description
	<p>OAuth2.</p> <ul style="list-style-type: none"> ■ Use credentials from incoming request: Default. This is known as “pass through” mode, in which the consumer includes an OAuth2 access token (a “Bearer” type token) in the request. Mediator then passes the access token unchanged to the native OAuth server. ■ Use specified token: In this mode, the consumer does not include an OAuth2 access token in the request. Instead, the provider generates an OAuth2 access token for each consumer, and Mediator stores the access tokens in Passman. When consumers send requests, Mediator obtains the OAuth2 access tokens from Passman and uses them to access the native services. Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the Show Token button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. <p>None. Select the following option:</p> <ul style="list-style-type: none"> ■ Invoke Service Anonymously: Does not authenticate requests.
HTTP Headers	<p>Specifies the HTTP headers used to authenticate the requests.</p> <ul style="list-style-type: none"> ■ Use Existing: Uses the HTTP headers that are contained in the requests. ■ Customize: Uses the HTTP headers that are specified in the Name and Value columns below.

Viewing or Editing Virtualized Services

Each tab on the detail page of a virtualized service represents a collection of attributes called a profile. You will only see the profiles for which you have View permission. The asset type Service has a unique set of profiles. However, your administrator can configure this asset type to display a customized set of profiles and attributes.

Important: *This is of specific relevance to REST and XML based virtual service assets.* Beginning with version 9.7, CentraSite supports the enhanced interface for REST and XML service assets (in contrast, earlier versions of CentraSite supported a standardized interface for REST and XML service assets). Note that the standardized REST and XML service interface that was implemented by versions of CentraSite prior to version 9.7 is not compatible with the enhanced interface that is implemented by current version of CentraSite. Documentation of the prior REST and XML service interface is available

to Software AG customers who have a current maintenance contract in Empower, Software AG's global extranet (<http://empower.softwareag.com/>).

If You Migrate REST/XML Services from a Pre-9.7 Release

If you have REST and XML services that were created prior to version 9.7, these REST services will continue to hold the old version's metadata in the enhanced REST service interface implemented by current version of CentraSite. Keep in mind that you can only view properties of the REST and XML services in CentraSite Control. You cannot configure properties of the REST or XML services using the CentraSite Control user interface (not even if you belong to the CentraSite Administrator role). You will only be able to configure the properties using the CentraSite Business UI.

When editing attributes, keep the following points in mind:

- If you are not the owner of the virtual service, you cannot edit the virtual service unless you have Modify permission on the virtual service (granted through either a role-based permission or an instance-level permission).
- When you view the details for the virtual service, you will only see profiles for which you have View permission. You will only be able to edit the profiles on which you have Modify permission.
- Some attributes accept only specific types of information. For example, if the virtual service includes a URL type attribute, you must supply a URL when you edit that attribute. Other attribute types that require a specific type of value include Date attributes and Email attributes.
- Some attributes are designed to be read-only and cannot be edited even if they appear in a virtual service on which you have Modify permission.

To view or edit the detail pages of a virtualized service

1. In CentraSite Control, go to **Asset Catalog > Browse Catalog**.
2. On the **Browse Catalog** page, perform a keyword search or advanced search to display the virtualized services. For procedures, see *CentraSite User's Guide*.
3. Locate the virtualized service whose details you want to view or edit and, from its context menu, select **Details**.

If you want to edit multiple services, mark the checkboxes of all desired services, and from the **Actions** menu, click **Details**. The detail page for each of the selected services is now displayed.

4. To edit an asset's **Name**, **Description** or user-defined version number, make sure the service is undeployed, then place the cursor in the appropriate field and modify the text as required.
5. To modify the extended attributes associated with the virtual service, do the following:
 - a. Select the profile that contains the attribute(s) that you want to modify.
 - b. Edit the attributes on the profile as necessary.

For a SOAP-based Virtual Service only: You can upload a new WSDL file, or update an existing WSDL file.

You attach a WSDL file to the catalog entry using the **Attach WSDL** command in the virtual service's **Actions** menu. If you are attaching a WSDL file to a service that already has a WSDL, the service name in the new WSDL must be identical to the service name in the existing one, or the process will fail.

- c. Repeat steps 5.a and 5.b for each profile that you want to edit.

Note: If at any time you want to abandon your unsaved edits, click **Close**. CentraSite will ask you if you want to save your edits. Click **No** to abandon your edits and return the virtual service's attributes to their previous settings.

6. When you have finished making your edits, click **Save**.

You can view a tooltip text for any attribute in a profile of the virtualized service's detail page by moving the cursor to the attribute name. The tooltip text gives a summary of the attribute's purpose. The tooltip text shown is the content of the attribute's `Description` field, as defined for the virtualized service in the Service type definition. See *CentraSite Administrator's Guide* for information on defining attributes for Service type.

The Summary Profile

The **Summary** profile displays general information about the service, including:

<u>For this service type...</u>	<u>The profile displays...</u>
Virtual Service	The WSDL file and the native service's endpoint(s).
Virtual REST Service	The schema file, the native service's endpoint(s) and the HTTP methods (GET, POST, PUT, DELETE or Use Context Variable) that the native service accepts in requests.
Virtual XML Service	The schema file and the native service's endpoint(s).

The Technical Details Profile

The **Technical Details** profile displays the following:

<u>For this virtualized service type...</u>	<u>The profile displays...</u>
Web service	■ The WSDL file of the Web service.

<u>For this virtualized service type...</u>	<u>The profile displays...</u>
REST service or XML service	<ul style="list-style-type: none"> ■ The native service's endpoint. ■ The schema file of the REST service or XML service. ■ The native service endpoint. ■ The MIME type of the data supported by the service. For a REST service, this is often application/xml or application/json type but can be any other valid MIME type. For an XML service, this is only application/xml type. ■ The set of operations supported by the service using HTTP methods (for example, POST, GET, PUT or DELETE). ■ The search string used by the server to find a set of matching resources.

The Specification Profile

You use the **Specification** profile to view or modify the following fields.

<u>Field</u>	<u>Description</u>
Functional Requirements	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library. For details about the Supporting Document Library, see the <i>CentraSite User's Guide</i> .
Non-functional Requirements	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library.
Error Messages and Codes	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library.
Usage Examples	You can attach a document by either pointing to a URL, or selecting a document from the Supporting

Field	Description
	Document Library. You can also upload a new document to the library.
Release Notes	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library.
Demo WSDL URL	You can view or modify the Demo WSDL URL.
Documentation URL	You can view or modify the Documentation URL.
Consumer WSDL	<p>If the built-in policy Consumer WSDL Generator has been activated, this field will display a <i>Consumer WSDL</i>. For information about this policy, see the <i>CentraSite User's Guide</i>.)</p> <p>A Consumer WSDL is created in addition to the virtual service's WSDL. The virtual service's WSDL will be used by Mediator. The Consumer WSDL can be used by the consumer (the user) to create a request for the service. The Consumer WSDL will contain WS-Security policies inline in the consumer WSDL as follows:</p> <ul style="list-style-type: none"> ■ If the virtual service contains WS-Security policies, then the WS-Security policies are included inline in the Consumer WSDL. Any WS-Security policies contained in the native service's WSDL are removed. ■ If the virtual service does <i>not</i> contain WS-Security policies, then the WS-Security policies contained in the native service's WSDL (if any) are included inline in the Consumer WSDL.

The Consumers Profile

The **Consumers** profile displays the list of users, applications and arbitrary assets that are registered to consume the virtualized service.

The Permissions Profile

You use the **Permissions** profile to set permissions for the service. For information, see the *CentraSite Administrator's Guide*.

Important: To set permissions on a virtualized service, you must have the Full instance-level permission on the native service from which the service was generated (or you must belong to a role that has the Manage Assets permission).

The Policies Profile

The **Policies** profile displays a list of all design-time policies and run-time policies that apply to the service. To view a policy's detail page, click the hyperlinked policy name.

The Deployment Profile

For instructions on deploying, undeploying, and redeploying virtualized services, see "[Deploying and Undeploying Virtualized Services to Targets](#)" on page 151.

The Performance Profile

The **Performance** profile displays the Key Performance Indicator (KPI) metrics that have been published for the service. You can filter the list by target and time period.

Note: Keep the following in mind:

- Ensure that Mediator is configured to collect and report run-time events to CentraSite, as described in *Administering webMethods Mediator*.
- Ensure that CentraSite is configured to receive run-time events from Mediator, as described in "[Run-Time Governance Reference](#)" on page 327.

Note: If you receive a Javascript error when trying to display the **Performance** profile, please install the latest versions of the Adobe Flash Player/Shockwave Player plug-ins on your Microsoft Internet Explorer.

To view KPI metrics

1. In CentraSite Control, display the details page of virtualized service. If you need procedures for this step, see "[Viewing or Editing Virtualized Services](#)" on page 120.
2. Open the **Performance** profile.
3. Use the **Switch to** button to switch between a tabular view of the metrics or a graphical view.
4. When viewing metrics in Tabular View, specify the following fields:

In this field...	Specify...
Select Target	A target to which the virtualized service is deployed, or select All to view the metrics of all targets to which the virtualized service is deployed.
Start Date/End Date	The time period from which to view the metrics.

5. Click **Search**.

The table displays metrics for all performance categories (Success Request Count, Total Request Count, Fault Count, and so on).

You can view a tooltip text for any attribute in a profile of the virtualized service's detail page by moving the cursor to the attribute name. The tooltip text gives a summary of the attribute's purpose. The tooltip text shown is the content of the attribute's `Description` field, as defined for the virtualized service in the Service type definition. See *CentraSite Administrator's Guide* for information on defining attributes for Service type.

The Events Profile

The virtualized service's **Events** profile displays the run-time events for the service. You can filter the list by target, event type and time period.

- Note:** Keep the following in mind:
- Ensure that Mediator is configured to collect and report run-time events to CentraSite, as described in *Administering webMethods Mediator*.
 - Ensure that CentraSite is configured to receive run-time events from Mediator, as described in "[Run-Time Governance Reference](#)" on page 327.

To view run-time events for a virtualized service

1. In CentraSite Control, display the virtualized service's detail page. For procedures, see the *CentraSite User's Guide*.
2. Filter the event list you want to generate as follows:

In this field...	Specify...
Target Type	The type of the target on which the event occurred.
Target	The target on which the event occurred, or select All Targets .

In this field...	Specify...
Event Type	A run-time event type, or select All Events .
Date Range	A range of dates from which to view the events. If you select a value for Date Range, then Start Date/Time and End Date/Time are ignored.
Start Date	Click the calendar to specify a start date and time.
End Date	Click the calendar to specify an end date and time.

- Click **Search**.
- The generated event list displays the following information:

Column	Description
Date/Time	The date/time of the event. Click this hyperlinked value to view the payload of the request/response.
Session ID	The SOAP invocation session ID of the event.
Event Type	The type of run-time event.
Service Name	The name of the service involved in the event.
Service Type	The service's type.
Target	The target on which the event occurred.
Target Type	The target's type.

To configure CentraSite to log run-time events, see the *CentraSite Administrator's Guide*.

Note: To view lists of all events for all virtualized services of a particular target (or for all targets), see "[Run-Time Governance Reference](#)" on page 327.

Revising Virtualized Services

For details, see "[Virtualized Services in CentraSite](#)" on page 67

Creating Run-Time Policies

You create run-time policies and apply them to virtual services in order to govern the virtual services' run-time execution.

A run-time policy is a sequence of actions that are carried out by a policy-enforcement point (PEP) when a consumer requests a particular service through the PEP. The actions in a run-time policy perform activities such as identifying/authenticating consumers, validating digital signatures and capturing performance metrics. You create run-time policies using CentraSite Control and store them in the CentraSite registry/repository.

When you create a run-time policy in CentraSite Control, you:

- Specify the PEP target type (for example, webMethods Mediator) to which you will deploy the virtual services and their policies.
- Add run-time actions to the policy and configure their parameters. CentraSite provides a set of built-in run-time actions.
- Apply the policy to the desired virtual services.
- Activate the policy.

The content is organized under the following sections:

Actions that Run-Time Policies can Execute

A run-time action is a single task that is included in a run-time policy and is evaluated by a policy-enforcement point (PEP). Actions in run-time policies perform tasks such as identifying/authenticating consumers and logging transaction activity. You specify actions when you define the policy. The PEP evaluates actions in the order in which they appear in the list of actions.

CentraSite provides *run-time action templates*. A run-time action template is a definition of an action that can be used in a run-time policy. Most action templates specify a set of parameters associated with a particular policy action. For example, when you configure the action that identifies consumers you specify an identifier (for example, an HTTP Authentication token) to identify the consumers who are trying to access the services. You can include multiple actions in a single policy.

Using the action templates, you can configure the following types of run-time actions:

■ **WS-SecurityPolicy 1.2 actions**

Mediator provides two kinds of actions that support WS-SecurityPolicy 1.2: authentication actions and XML security actions.

- You use the authentication actions to verify that the service consumer has the proper credentials to access a virtual service. You can authenticate consumers by their WSS X.509 certificates, WSS Username tokens or WSS SAML tokens.

- You use the XML security actions to provide confidentiality (through encryption) and integrity (through signatures) for request and response messages.

■ **Monitoring actions**

Mediator provides the following run-time monitoring actions:

- The “Monitor Service Performance” action, which monitors a user-specified set of run-time performance conditions for a virtual service, and sends alerts to a specified destination when these conditions are violated.
- The “Monitor Service Level Agreement” action, which provides the same functionality as “Monitor Service Performance”, but this action is different because it enables you to monitor a virtual service's run-time performance for particular consumers. You configure this action to define a *Service Level Agreement (SLA)*, which is set of conditions that defines the level of performance that a specified consumer should expect from a service.
- The “Throttling Traffic Optimization” action (not available in Mediator versions below 9.0), which limits the number of service invocations allowed during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated. You can use this action to avoid overloading the back-end services and their infrastructure, to limit specific consumers in terms of resource usage, etc.

■ **Additional actions**

Mediator provides the following actions, which you can use in conjunction with the actions above.

- “Identify Consumer”, which you use in conjunction with an authentication action (“Require WSS Username Token”, “Require WSS X.509 Token” or “Require HTTP Basic Authentication”). Alternatively, you can use this action alone to identify consumers only by host name or IP address.
- “Require HTTP Basic Authentication”, which uses HTTP basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header against the Integration Server.
- “Authorize User”, which authorizes consumers against a list of users and/or a list of groups registered in the Integration Server on which Mediator is running. You use this action in conjunction with an authentication action (“Require WSS Username Token”, “Require WSS SAML Token” or *Require HTTP Basic Authentication*).
- “Authorize Against Registered Consumers”, which authorizes consumer applications against all Application assets that are registered in CentraSite as consumers for the service.
- “Log Invocation”, which logs request/response payloads to a destination you specify.
- “Validate Schema”, which validates all XML request and/or response messages against an XML schema referenced in the WSDL.

For detailed descriptions of these actions, see ["Built-In Run-Time Actions Reference for Virtual Services"](#) on page 347.

Who Can Create and Manage Run-Time Policies?

To create and manage run-time policies, you must belong to a role that includes the Manage Runtime Policies permission (or the Manage System-wide Runtime Policies permission).

By default, the following predefined roles include the Manage Runtime Policies permission:

- Organization Administrator
- Policy Administrator

By default, the following predefined roles include the Manage System-wide Runtime Policies permission:

- CentraSite Administrator
- Operations Administrator

Note: For more information about roles and permissions, see *Getting Started with CentraSite*.

Creating a Run-Time Policy

To create a run-time policy in CentraSite, you perform the following high-level steps:

1. Create a new run-time policy. During this step, you select the actions you want the policy to execute, and you assign values to the action parameters.
2. Allow other users to view, edit and/or delete this policy by assigning permissions to those users. For procedures, see ["Setting Permissions on a Run-Time Policy"](#) on page 134.
3. Activate the policy. During this step, you make the new policy ready to deploy to a PEP. For procedures, see ["Activating a Run-Time Policy"](#) on page 135.

Perform these steps to create a run-time policy and save it to CentraSite.

To create a run-time policy

1. In CentraSite Control, go to **Policies > Run-Time**.
2. Click **Add Policy**.
3. In the **Policy Information** panel, specify the following fields:

In this field...	Specify...
Name	A name for the new policy. A policy name can contain any character (including spaces).
Description	<i>Optional.</i> A description for the new policy. This description appears when a user displays a list of policies in the user interface.
Version	<p><i>Optional.</i> A version identifier for the new policy.</p> <p>Note: The version identifier does not need to be numeric.</p> <p>Examples:</p> <pre>0.0a 1.0.0 (beta) Pre-release 001 V1-2007.04.30</pre> <p>Note: The version identifier you enter here is the policy's public, user-assigned version identifier. CentraSite also maintains an internal, system-assigned version number for the policy. For more information about user-assigned and system-assigned version identifiers, see "System-Assigned vs. User-Assigned Version Identifiers" on page 145.</p>

- In the **Scope** panel, specify the following fields. *Scope* refers to the set of properties that determine the target type, organization and asset type to which the policy applies.

In this field...	Specify...
Target Type	The target type to which the policy will be deployed. Select webMethods Integration Server (that is, the webMethods Mediator gateway type).
Organization	<p>The organization to which the policy applies. Choose All if you want to apply the policy to the specified services in all organizations.</p> <p>Important: Once you create a policy, its organizational scope is fixed and cannot be changed. That is, if you create a policy whose scope is specific to organization ABC, you cannot change its scope to make it system-wide or switch it to another organization. You must create a new policy and set its organizational scope as needed.</p>

In this field...	Specify...
Asset Types	<p>The type of asset to which this policy applies. Choose one of the following:</p> <ul style="list-style-type: none"> ■ Service ■ XML Service ■ REST Service ■ Virtual Service ■ Virtual XML Service ■ Virtual REST Service <p>Note: CentraSite does not provide out-of-the-box policy-enforcement for web services.</p>

5. In the **Apply Policy to Services that Meet the Following Criteria** panel, specify criteria that identify the virtual services to which the policy applies.

To target a policy for a particular set of virtual services, you refine the policy's scope by specifying additional selection criteria based on the virtual service's Name, Description or Classification properties.

- a. Choose an attribute (Name, Description or Classification) that identifies the services to which the policy applies.
- b. Choose an operator for the attribute (if applicable).
- c. Specify a value for the attribute (if applicable). Values are case-sensitive.
- d. If you need to specify multiple values or attributes, use the plus button to add multiple rows. For example, for the Classification attribute you might choose multiple Taxonomy names. If you specify multiple criteria, they are connected by the AND operator.

After you save the policy, you will see the generated list of services displayed on the Policy Detail page's **Services** profile.

Note: Keep the following in mind:

- If you specify no criteria, the policy will apply to all virtual services.
- You can specify only *one* "Name Equals <value>" condition. However, you can specify multiple "Name Contains <value>" or "Name Starts With <value>" conditions.

Caution: CentraSite checks for policy conflicts when you deploy a virtual service to Mediator. If the service has only one policy applied to it (the policy you are applying here), that policy is deployed to Mediator, and Mediator executes the policy's run-time actions in the order in which

they appear in the policy. However, if the service already has additional policies applied to it, a policy conflict might occur, which might cause unintended consequences. CentraSite will inform you of policy conflicts. For information about how Mediator evaluates actions (and how to avoid policy conflicts), see "[Action Evaluation Order and Dependencies](#)" on page 351.

6. Click **Next**.
7. In the **Available Actions** dialog, select the built-in actions that you want to include in the policy. Keep the following points in mind when you select the actions for the policy:
 - If you are using webMethods Mediator as your PEP, you must include the "Identify Consumer" built-in action (and optionally other identification actions) in order to identify or authenticate consumers. For common usage cases of identification actions, see "[Usage Cases for Identifying/Authenticating Consumers](#)" on page 355.
 - Ensure that the actions in the **Selected Actions** list appear in the order in which you want them to run when the policy is enforced. If necessary, use the control buttons on the right side of the list to place them in the correct order.
8. Click **Finish** to save the new (as yet incomplete) policy. The Runtime Policy Detail page is displayed, showing details of the policy you just created.
9. Specify parameter values for each of the policy's actions as follows:
 - a. In the **Actions** profile, choose the action whose parameters you want to set.
 - b. In the **Edit Action Parameters** page, set the parameters as necessary and click **Save**. Required parameters are marked with an asterisk. For detailed information about the parameters, see "[Built-In Run-Time Actions Reference for Virtual Services](#)" on page 347.
 - c. When you have finished setting the parameters of all actions in the list, click **Save** and then **Close**. The icons next to the actions in the **Parameters Set** column will indicate whether the action parameters have been set.

Icon	Description
	The action has required input parameters that have not yet been set.
	All of the action's required input parameters have been set.

Note: This icon automatically appears for actions that have no input parameters.

10. Complete the policy by doing the following:
 - a. If you want to allow other users to view, edit and/or delete this policy, go to the **Policy Detail** page, select the **Permissions** profile, and assign permissions to

those users. You will not see this profile unless you belong to a role that has the Manage Runtime Policies permission. For procedures, see the *CentraSite Administrator's Guide*.

- b. Activate the policy when you are ready to put it into effect. For procedures, see ["Activating a Run-Time Policy" on page 135](#).

Setting Permissions on a Run-Time Policy

If you want to permit other users to manage (that is, view, edit, and delete) run-time policies, you do so by adding to the policy's instance-level permission settings. A policy has the following types of instance-level permission settings.

Permission	Description
View	Enables users to see the policy in their policy list and view details for the policy.
Modify	Enables users to view and modify the properties of a policy (including the policy's scope and action list).
Full	Enables users to view, modify or delete the policy.

Who Can Set Permissions on a Run-Time Policy?

To set permissions on a policy, you must belong to a role that has Manage Runtime Policies permission or the Manage System-wide Runtime Policies permission.

Assigning Permissions to a Run-Time Policy

You can assign view, edit, and delete permissions to any individual user or group defined in CentraSite.

Note: If you give a user permission to view, edit or delete a policy, and you want that user to be able to perform these operations using CentraSite Control, make sure that the user belongs to a role that also has the Use the Policy UI permission.

To assign permissions to a policy

If you are modifying the permissions of an active policy, you must first deactivate the policy.

1. Display the Policy Detail page for the policy that you want to activate. If you need procedures for this step, see ["Modifying a Run-Time Policy" on page 139](#).
2. Select the **Permissions** profile.

3. To add a new a new user or group to the list, do the following:
 - a. Choose **Add Users / Groups**.
 - b. Select the users and/or groups to which you want to assign permissions.

<u>If you type...</u>	<u>CentraSite will return...</u>
b%	Groups and user names that start with the letter b.
b	Groups and user names that contain the letter b.
%b	Groups and user names that end with b.
%	All groups and user names.

If you want to filter the user list, specify a pattern-matching string in the **Search** field. The pattern-matching string can consist of one or more characters and/or the % wildcard symbol. The string is matched against group names and user IDs.

- c. Choose **OK**.
4. Use the **Full**, **Modify** and **View** check boxes to assign specific permissions to the users and groups in the list.
5. To remove an entry from the list, select the check box beside the group or user name and click **Delete**.
6. Click **Save** to save your settings.

Activating a Run-Time Policy

When you activate a run-time policy, CentraSite applies it to the services you specified in the policy.

To activate a policy, you change the policy's lifecycle state to the `Productive` state. This state change executes CentraSite's *Automatic Policy Activation* policy.

Note: The *Automatic Policy Activation* policy is a hidden system policy. You cannot edit or delete this policy.

When you activate a policy, be aware that:

- You will not be allowed to activate the policy unless all of its parameters have been set. When you switch the policy to the `Productive` state, CentraSite executes the *Validate Policy Activation* policy. This policy will not allow you to switch a policy to the `Productive` state if the policy's parameters have not yet been set.
- Some organizations require an approval to activate a policy. If your organization has an approval action associated with the activation of a policy, CentraSite will not

activate the policy until the required approvals are obtained. For more information about approval actions, see the *CentraSite User's Guide*.

- To activate a policy, you must have permission to change the policy to the Productive state.
- To successfully change a policy to the `Productive` state, you must also have the Modify permission on all virtual type services to which the policy is applied.

To determine whether a policy is active or inactive, examine the policy's **Active** indicator on the **Policies > Run-Time** page. The icon in the **Active** column indicates the policy's activation state as follows:

Icon	Description
	Policy is active.
	Policy is inactive.

The activation state of a policy is also reported next to the **State** field in the Run-Time Policy Details page.

To activate a policy

1. Display the Run-Time Policy Details page for the policy you want to activate. For procedures for this step, see ["Modifying a Run-Time Policy" on page 139](#).
2. Examine the **Actions** profile and verify that all of the actions on this profile display the green checkmark icon in the **Parameters Set** column. If any of the actions display the red circle icon in this column, set their parameters before you continue. For information about setting action parameters, see ["Modifying Action Parameters" on page 142](#).
3. In the **Policy Information** panel, click the **Change State** button. (If you do not see the **Change State** button, it is probably because you do not have permission to change the lifecycle state of a policy.)
4. In the **Change Lifecycle State** dialog box, select the **Productive** lifecycle state and click **OK**.
5. Examine the **State** field in the **Policy Information** panel to verify that the policy's state has been changed.

If this state change requires approval, the **State** field will indicate that the policy is in the `pending` mode. CentraSite will automatically switch the policy to the requested state (and activate the policy) after all the necessary approvals have been obtained. For information about checking the status of objects that you have submitted for approval, see the *CentraSite User's Guide*.

Note: While the policy is in pending mode, it cannot be edited.

6. After you activate the policy, users with the proper permissions will deploy the services to your PEP. At that time, CentraSite will automatically validate the service's policies (for example, check for policy conflicts or other violations). For information about how Mediator evaluates actions (and how to avoid policy conflicts), see ["Action Evaluation Order and Dependencies" on page 351](#).

Deactivating a Run-Time Policy

You usually deactivate a policy if you want to edit the policy (for example, to modify the scope of a policy or change its action list).

To deactivate a policy, you change the policy to the Suspended state. Switching the policy to this state triggers the *Automatic Policy Deactivation* policy, which deactivates the policy. (Switching the policy to the Retired state also deactivates the policy, but you do not want to switch a policy to this state unless you intend to deactivate it permanently. After you place a policy in the Retired state, you cannot reactivate it.)

To deactivate a policy, you must have permission to change the policy to the Suspended state.

To deactivate a policy

1. Display the Run-Time Policy Details page for the policy you want to activate. If you need procedures for this step, see ["Modifying a Run-Time Policy" on page 139](#).
2. In the **Policy Information** panel, click the **Change State** button. (If you do not see the **Change State** button, it is probably because you do not have permission to change the lifecycle state of a policy.)
3. In the **Change Lifecycle State** dialog box, select the **Suspended** state (to deactivate it temporarily) or the **Retired** state (to deactivate it permanently), then click **OK**

Viewing the Run-Time Policy List

The Run-Time Policies page displays a list of all run-time policies residing on the CentraSite server.

To view the policy list

1. In CentraSite Control, go to **Policies > Run-Time**.
2. By default, all of the available policies are displayed.

If you want to filter the list to see just a subset of the available policies, type a partial string in the **Search** field. CentraSite applies the filter to the **Name** column. The **Search** field is a type-ahead field, so as soon as you enter any characters, the display will be updated to show only those policies whose name contains the specified characters. The wildcard character % is supported.

3. You can sort the list by type of asset. To specify the sorting order, choose either **Web Service**, **REST Service**, **XML Service**, **Virtual Service**, **Virtual REST Service** or **Virtual XML Service** from the drop-down list labeled **Browse by**.

The Run-Time Policies page provides the following information about each policy.

Note: Only the first six columns described below are displayed in this list by default. To display the additional columns, click the **Select Columns** button.

<u>Column</u>	<u>Description</u>						
Name	The name assigned to the policy.						
Description	Additional comments or descriptive information about the policy.						
System Version	The automatically generated system-assigned version identifier for the policy. For more information about system-assigned version identifiers, see, " System-Assigned vs. User-Assigned Version Identifiers " on page 145.						
Organization	The organization to which the policy applies.						
	<table border="1"> <thead> <tr> <th><u>This value...</u></th> <th><u>Indicates that...</u></th> </tr> </thead> <tbody> <tr> <td>All</td> <td>The policy is system-wide and applies to all organizations.</td> </tr> <tr> <td><i>OrgName</i></td> <td>The policy applies to the specified organization.</td> </tr> </tbody> </table>	<u>This value...</u>	<u>Indicates that...</u>	All	The policy is system-wide and applies to all organizations.	<i>OrgName</i>	The policy applies to the specified organization.
<u>This value...</u>	<u>Indicates that...</u>						
All	The policy is system-wide and applies to all organizations.						
<i>OrgName</i>	The policy applies to the specified organization.						
State	The policy's current lifecycle state.						
Active	The policy's current enforcement state.						
	<table border="1"> <thead> <tr> <th><u>Icon</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td></td> <td>The policy is active. CentraSite enforces this policy when events within the scope of the policy occur.</td> </tr> <tr> <td></td> <td>The policy is inactive. Inactive policies exist in the registry, but they are not enforced.</td> </tr> </tbody> </table>	<u>Icon</u>	<u>Description</u>		The policy is active. CentraSite enforces this policy when events within the scope of the policy occur.		The policy is inactive. Inactive policies exist in the registry, but they are not enforced.
<u>Icon</u>	<u>Description</u>						
	The policy is active. CentraSite enforces this policy when events within the scope of the policy occur.						
	The policy is inactive. Inactive policies exist in the registry, but they are not enforced.						

Column	Description
Version	The user-assigned version identifier for the policy. For more information about user-assigned version identifiers, see, " System-Assigned vs. User-Assigned Version Identifiers " on page 145.
Owner	The user to which the policy belongs.

Modifying a Run-Time Policy

Viewing or Changing a Run-Time Policy

You use the Run-Time Policy Details page to examine and/or edit the properties for a policy. When editing a policy, keep the following points in mind:

- To edit a policy, you must have Modify permission on the policy. If your user account belongs to a role that has the Manage Runtime Policies permission for an organization, you automatically have permission to modify all of the policies in that organization. If your user account belongs to a role that has the Manage System-Wide Runtime Policies permission, you have permission to edit any system-wide policy on the server.
- You cannot make changes to a run-time policy while it is active. To make changes to an active policy, you must switch the policy to the Suspended state (to deactivate it), update the policy and then switch it back to the Productive state (to reactivate it). For information about deactivating and activating a policy, see "[Deactivating a Run-Time Policy](#)" on page 137 and "[Activating a Run-Time Policy](#)" on page 135, respectively.

To edit a policy

1. In CentraSite Control, go to **Policies > Run-Time** to display the policy list.
2. Locate the policy whose details you want to view or edit and choose **Details** from its context menu.
3. Examine or modify the properties on the **Policy Detail** page as necessary.

Field or Profile	Description
Name	The name of the policy. A policy name can contain any character (including spaces).
Description	<i>Optional.</i> Additional comments about the policy.
Version	The user-defined version number to be assigned to the new version. We recommend that you update the version number

Field or Profile	Description						
	anytime you make significant modifications to a policy. The usage of this field is the same as described in "Creating a Run-Time Policy" on page 130 .						
State	The policy's current lifecycle state (e.g., New, Productive, Suspended, Retired). This field also displays an icon that indicates the activation state of the policy:						
	<table border="1"> <thead> <tr> <th>Icon</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td></td> <td>The policy is active (that is, ready to be deployed to a PEP).</td> </tr> <tr> <td></td> <td>The policy is inactive.</td> </tr> </tbody> </table>	Icon	Description		The policy is active (that is, ready to be deployed to a PEP).		The policy is inactive.
Icon	Description						
	The policy is active (that is, ready to be deployed to a PEP).						
	The policy is inactive.						
Organization	The organization to which the policy belongs.						
Owner	The user who created the policy.						
System Version	The automatically-generated system version identifier for the policy. For more information, see "Creating a Run-Time Policy" on page 130 .						
Actions profile	The settings in this profile specify the actions that the PEP will execute when the policy is enforced. For more information about the properties on this profile, see "Creating a Run-Time Policy" on page 130 .						
Scope profile	The settings in this profile determine the services to which the policy is applied. For more information about the properties on this profile, see Modifying the Scope of a Run-Time Action .						
Services profile	Displays the list of Web services and/or virtual services to which the policy applies. For more information, see Viewing the List of Services To Which a Run-Time Policy Applies .						
Permissions profile	The settings in this profile identify which users can view, edit and/or delete the policy. For more information about the properties on this profile, see "Setting Permissions on a Run-Time Policy" on page 134 .						

- If you edited the settings on the **Run-Time Policy Details** page, click **Save** to save the updated policy.

5. If you deactivated the policy in order to edit it, activate the policy as described in ["Activating a Run-Time Policy" on page 135](#).

You can view the details page of multiple policies as follows:

To view the details page of multiple policies

1. In CentraSite Control, go to **Policies > Run-Time** to display the policy list.
2. Locate the policies whose details you want to view, and mark their checkboxes.
3. In the **Actions** menu, choose **Details**.

The details page of each of the selected policies will now be displayed.

Modifying Actions for a Run-Time Policy

Modifying the Actions List

Use the procedure in this section to modify the actions list.

To modify the actions list

1. Display the **Policy Detail** page for the policy whose actions you want to edit. For procedures for this step, see ["Modifying a Run-Time Policy" on page 139](#).
2. If the policy is active, deactivate it. You cannot modify an active policy. For procedures for this step, see ["Deactivating a Run-Time Policy" on page 137](#).
3. Select the **Actions** profile to display the list of actions associated with the policy.
4. To add actions to, delete actions from, or modify the order of actions in the list, do the following:
 - a. Click **Edit Actions List**.
 - b. Use the controls in the **Edit Assigned Actions List** dialog box to add actions to the list and/or delete actions from the list.

When editing the list of actions, keep the following point in mind:

- If you are using webMethods Mediator as your PEP, you must include the built-in run-time action "Identify Consumer" (and possibly other authorization/identification actions) in order to identify or authenticate consumers. For common usage cases of identification actions, see ["Usage Cases for Identifying/Authenticating Consumers" on page 355](#).
 - If necessary, you can click **Previous** to return to the previous page and modify the information in the **Policy Information** panel.
- c. Make sure that the actions in the **Selected Actions** list appear in the order that you want them to run when the policy is enforced. If necessary, use the control buttons on the right side of the list to place them in the correct order.
 - d. Click **OK** to save the modified list.

5. Use the procedure in [Modifying Action Parameters](#) to specify parameter values for any new actions that you might have added to the list, or to make any necessary updates to the parameter values for existing actions.
6. When the action list is complete and you have configured all of the input parameters for the actions correctly, click **Save** to save the updated policy.
7. If you deactivated the policy in order to edit it, activate the policy as described in ["Activating a Run-Time Policy" on page 135](#).

Modifying Action Parameters

Policy actions have parameters that you set to configure the behavior of the action at enforcement time. When you display the **Actions** profile on the Policy Detail page, the icon in the **Parameters Set** column indicates whether an action has input parameters that need to be set.

This icon...	Indicates that...
	The action has required input parameters that have not yet been set.
	All of the action's required input parameters have been set.

Note: This icon automatically appears for actions that have no input parameters.

Until the green check mark icon appears for all actions, you will not be able to activate the policy (if it is inactive) or save the policy (if it is already active).

To modify action parameters

1. Display the **Policy Detail** page for the policy whose actions you want to edit. For procedures for this step, see ["Modifying a Run-Time Policy" on page 139](#).
2. If the policy is active, deactivate it. You cannot modify an active policy. For procedures for this step, see ["Deactivating a Run-Time Policy" on page 137](#).
3. In the **Actions** profile do the following for each action in the list.
 - a. Choose the action whose parameters you want to examine or set.
 - b. In the **Edit Action Parameters** page, set the parameters as necessary.

Note: Required parameters are marked with an asterisk. For more information about built-in actions and their parameters, see ["Built-In Run-Time Actions Reference for Virtual Services" on page 347](#).
 - c. Click **Save** to save the parameter settings.
4. After you configure the parameters for all of the actions in the list, click **Save** to save the updated policy.

- If you deactivated the policy in order to edit it, activate the policy as described in ["Activating a Run-Time Policy" on page 135](#).

Modifying the Scope of a Run-Time Action

You can use the **Scope** profile on the Run-Time Policy Detail page to modify a policy's scope.

Note: Once you create a policy, its organizational scope is fixed and cannot be changed. That is, if you create a policy whose scope is specific to organization ABC, you cannot change its scope to make it system-wide or switch it to another organization. You must create a new policy and set its organizational scope as needed.

To modify the scope of a run-time policy

- Display the Policy Detail page for the policy that you want to edit. For procedures for this step, see ["Modifying a Run-Time Policy" on page 139](#).
- If the policy is active, deactivate it. You cannot modify an active policy. For procedures for this step, see ["Deactivating a Run-Time Policy" on page 137](#).
- In the **Scope** profile, you can modify the following fields.

Field or Profile	Description
Target Type	The target type to which the policy will be deployed (for example, webMethods Integration Server).
Asset Type	The type(s) of assets to which this policy applies. Note: CentraSite does not provide out-of-the-box policy-enforcement for web services.
Organization	Read-only field. You cannot change the policy's organization after the policy has been created.
Apply Policy to Services that Meet the Following Criteria	The services to which the policy applies. <ul style="list-style-type: none"> ■ Choose an attribute (for example, Name, Description, Classification) that identifies the services to which the policy applies. ■ Choose an appropriate operator. ■ Specify a value. ■ If you need to specify multiple criteria, use the plus button to add multiple rows. For example, for the Classification attribute you might choose multiple

Field or Profile	Description
	<p>Taxonomy names. After you save the policy, you will see the generated list of services displayed on the Policy Detail page's Services profile.</p> <p>Note: If you specify multiple criteria, the criteria are connected with the AND operator. That means that <i>all</i> criteria must be satisfied, in order for the policy to apply to any of the services. If not all the criteria are satisfied, the policy will not apply to any of the services.</p>

4. Click **Save** to save the modified policy.
5. If you deactivated the policy in order to edit it, activate the policy as described in ["Activating a Run-Time Policy" on page 135](#).

Viewing the List of Services To Which a Run-Time Policy Applies

Use the following procedure to view the list of services to which a run-time policy applies.

Important: The list only includes services that are in the “deployable state” (that is, services whose deployment state has been enabled by the Change Deployment State action in a design-time policy). Services that are within the scope of the policy, but have not yet been made “deployable”, do not appear in this list.

To view the list of services

1. Display the Policy Detail page for the policy that you want to view. For procedures for this step, see ["Viewing or Changing a Run-Time Policy" on page 139](#).
2. Select the **Services** profile to view the list of virtual services and/or Web services that was generated based on the criteria you specified in the **Scope** profile.
3. To view details of a service, click its hyperlinked name.

Note: To add or remove services to or from the list, return to the **Scope** profile and change the criteria in the **Apply Policy to Services that meet the following Criteria** panel. For more information, see the procedure in [Modifying the Scope of a Run-Time Action](#).

Deleting a Run-Time Policy

You delete a policy to remove it from CentraSite permanently.

CentraSite is installed with a system-wide policy called *Check State Validation Policy for Policy*. This policy will not allow you to delete a policy unless the policy is in the New or Retired state.

To delete a policy

1. In the CentraSite Control, go to **Policies > Run-Time** to display the policy list.
2. Ensure that the policy is deactivated (see "[Deactivating a Run-Time Policy](#)" on page 137).
3. Enable the checkbox next to the name of the policy that you want to delete.
4. Click **Delete**.

You can delete multiple policies in a single step. The rules described above for deleting a single policy apply also when deleting multiple policies.

To delete multiple policies in a single operation

1. In CentraSite Control, go to **Policies > Run-Time** to display the policy list.
2. Mark the checkboxes of the policies that you want to delete.
3. From the **Actions** menu, choose **Delete**.

System-Assigned vs. User-Assigned Version Identifiers

CentraSite maintains two version identifiers for a policy: a *system-assigned identifier* and a *user-assigned identifier*.

- The system-assigned identifier is a version number that CentraSite maintains for its own internal use. CentraSite automatically assigns this identifier to a policy when the policy is created. You cannot delete it or modify it. A policy's system-assigned identifier is numeric.

A policy's system-assigned version number is shown in the **System Version** column on the Run-Time Policies page and in the **System Version** field of the policy's detail page.

- The user-assigned identifier is an optional identifier that you can assign to a distinguish a specific version of a policy. This identifier does not need to be numeric. For example, you might use a value such as "V2.a (beta)" to identify a version.

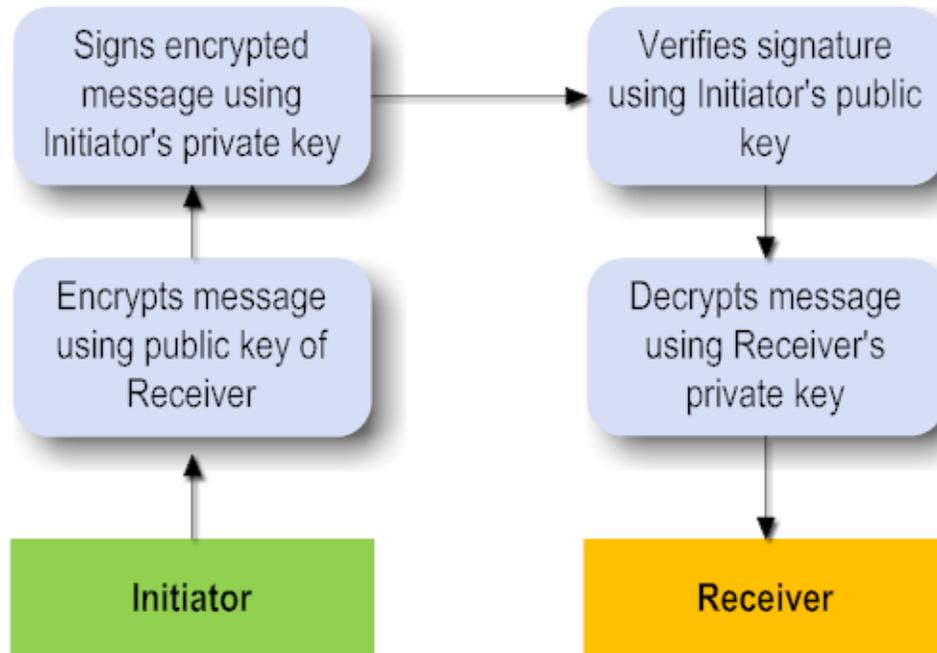
A policy's user-assigned version number is shown in the **Version** column on the Run-Time Policies page and in the **Version** field of the policy's detail page.

Asymmetric Binding Configuration

WS-SecurityPolicy specification deals with three types of Security Bindings. A security binding determines how the message transfer is to be done between the recipient and the initiator.

Asymmetric Binding

Asymmetric Binding is used when both the Initiator and the Recipient possess public and private keys. The message transfer takes place using Public Key Infrastructure.



An Asymmetric Binding element in the WSDL looks like this:

```

<sp:AsymmetricBinding
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
  <wsp:Policy>
    <sp:InitiatorToken>
    </sp:InitiatorToken>
    <sp:RecipientToken>
    </sp:RecipientToken>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:TripleDesRsa15/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:Layout>
      <wsp:Policy>
        <sp:Strict/>
      </wsp:Policy>
    </sp:Layout>
    <sp:IncludeTimestamp/>
    <sp:OnlySignEntireHeadersAndBody/>
  </wsp:Policy>
</sp:AsymmetricBinding>
  
```

The following run-time actions that support WS-Security policies use a common Asymmetric Binding element:

- The “Require Encryption” action.

- The “Require Signing” action.
- The “Require WSS SAML Token” action.
- The “Require WSS X.509 Token” action.

The Asymmetric Binding Components

The components of a security binding are as described below.

Initiator Token Inclusion

The value of Initiator Token Inclusion specifies how to include the initiator token during message exchange from Initiator to Recipient or Recipient to Initiator.

Value	Description
Always	Always include the token.
AlwaysToRecipient	Include the token in all message exchanges from the Initiator to Recipient.
AlwaysToInitiator	Include the token in all message exchanges from the Recipient to Initiator.
Once	Only once.
Never	Do not include the token in any communications.

Recipient Token Inclusion

The value of Recipient Token Inclusion specifies how to include the Recipient token during message exchange from Initiator to Recipient or Recipient to Initiator. It takes the same values as Initiator Token Inclusion above.

Algorithm Suite

The value of Algorithm Suite specifies the algorithm suite to be used for this asymmetric binding. The possible algorithms supported are:

- Basic256
- Basic192
- Basic128
- TripleDes
- Basic256Rsa15

- Basic192Rsa15
- Basic128Rsa15
- TripleDesRsa15
- Basic256Sha256
- Basic192Sha256
- Basic128Sha256
- TripleDesSha256
- Basic256Sha256Rsa15
- Basic192Sha256Rsa15
- Basic128Sha256Rsa15

Layout

Layout describes the way information is added to the message header. The possible values are:

Value	Description
Strict	Items are added to the security header in a principle of "declare before use".
Lax	Items are added to the security header in any order that conforms to WSS: SOAP Message Security.
LaxTsFirst	Same as Lax except that the first item in the security header <i>must</i> be a <code>wsse:Timestamp</code> . Note that the <code>wsse:Timestamp</code> property <i>must</i> also be set to true in this case.
LaxTsLast	Same as Lax except that the last item in the security header <i>must</i> be a <code>wsse:Timestamp</code> . Note that the <code>wsse:Timestamp</code> property <i>must</i> also be set to true in this case.

The Asymmetric Binding Configuration Command Tool

You can change the Asymmetric Binding configuration by executing the following commands in the command line interface `CentraSiteCommand.cmd` (Windows) or `CentraSiteCommand.sh` (UNIX) of CentraSite. The tool is located in `<CentraSiteInstallDir>/utilities`.

If you start this command line tool with no parameters, you receive a help text summarizing the required input parameters.

The parameters of the command are case-sensitive, so for example the parameter `-url` must be specified as shown and not as `-URL`.

The configuration is maintained in the form of an XML file which is loaded with default values.

```
<?xml version="1.0" encoding="UTF-8" ?>
<AsymmetricBindingConfiguration>
<initiatorTokenInclusion>AlwaysToRecipient</initiatorTokenInclusion>
<recipientTokenInclusion>Never</recipientTokenInclusion>
<algorithmSuite>TripleDesRsa15</algorithmSuite>
<layout>Strict</layout>
</AsymmetricBindingConfiguration>
```

The commands include:

- [get Asymmetric Binding](#)
- [set Asymmetric Binding](#)
- [remove Asymmetric Binding](#)

get Asymmetric Binding

Use this command to obtain the current Asymmetric binding configuration values. Use a command of the following format:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd get Asymmetric
Binding [-url <CENTRASITE-URL>] -user <USER-ID>-password <PASSWORD>
```

The following table describes the complete set of input parameters that you can use with the `get Asymmetric Binding` utility:

Input Parameter	Description
<code>-url</code>	The fully qualified URL (<code>http://localhost:53307/CentraSite/CentraSite</code>) for the CentraSite registry/repository.
<code>-user</code>	The user ID of a user who has the CentraSite Administrator role.
<code>-password</code>	The password of the user identified by the parameter <code>-user</code> .

For example:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd get Asymmetric
Binding [-url "http://localhost:53307/CentraSite/CentraSite"] -user
"Administrator" -password "manage"
```

set Asymmetric Binding

Use this command to change the values in the CentraSite registry/repository.

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd set
Asymmetric Binding [-url <CENTRASITE-URL>] -user <USER-ID> -
password [-initiatorTokenInclusion <INITIATOR-TOKEN-INCLUSION>] [-
recieipientTokenInclusion <RECIPIENT-TOKEN-INCLUSION>] [-algorithmSuite
<ALGORITHM-SUITE>] [-layout <LAYOUT>]
```

The following table describes the complete set of input parameters that you can use with the set `Asymmetric Binding` utility:

<u>Input Parameter</u>	<u>Description</u>
<code>-url</code>	The fully qualified URL (<code>http://localhost:53307/CentraSite/CentraSite</code>) for the CentraSite registry/repository.
<code>-user</code>	The user ID of a user who has the CentraSite Administrator role.
<code>-password</code>	The password of the user identified by the parameter <code>-user</code> .
<code>-initiatorTokenInclusion</code>	Inclusion value for the Initiator Token. *
<code>-recipientTokenInclusion</code>	Inclusion value for the Recipient Token. *
<code>-algorithmSuite</code>	The algorithm to be used. *
<code>-layout</code>	The layout to be used. *

* At least one of the following parameters is required: `initiatorTokenInclusion`, `recipientTokenInclusion`, `algorithmSuite` and `layout`.

remove Asymmetric Binding

Use this command to remove the Asymmetric Binding configuration. It has the same parameters as the `get Asymmetric Binding` command:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd remove
Asymmetric Binding [-url <CENTRASITE-URL>] -user <USER-ID> -password
<PASSWORD>
```

The following table describes the complete set of input parameters that you can use with the `remove Asymmetric Binding` utility:

Input Parameter	Description
-url	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
-user	The user ID of a user who has the CentraSite Administrator role.
-password	The password of the user identified by the parameter -user.

For example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd remove  
Asymmetric Binding [-url "http://localhost:53307/CentraSite/CentraSite"]  
-user "Administrator" -password "manage"
```

Deploying and Undeploying Virtualized Services to Targets

This section describes how to deploy, undeploy and redeploy virtualized services to webMethods Mediator gateways.

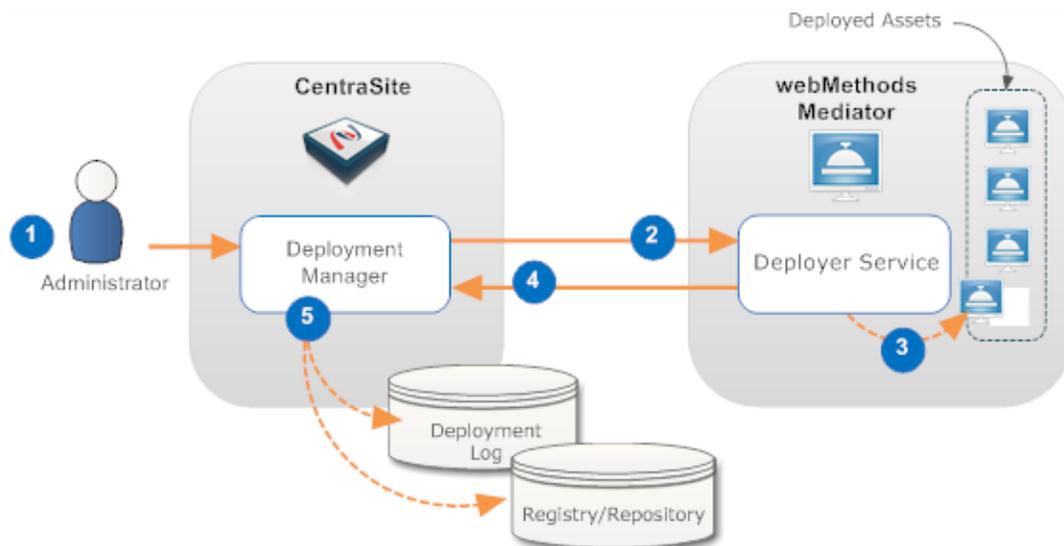
The Synchronous Deployment Model

Deployment refers to the process you use to send virtual services, virtual XML services, and virtual REST services to the policy-enforcement points (PEPs) on which they are to be used for run-time governance. Instructions throughout the remainder of this guide use the term “virtualized service” when referring to all three virtual types in general.

Note: This section provides information about deploying virtualized services to webMethods Mediator. If you are using a different kind of PEP, refer to its documentation for deployment procedures and information.

As shown in the following diagram, the deployment process is initiated from CentraSite and is carried out by the “deployer” service on Mediator.

The Deployment Process



Step	Description
1	An administrator initiates the deployment by selecting the assets that are to be deployed and specifies to which Mediator they are to be deployed.
2	The Deployment Manager on CentraSite prepares the asset for deployment (the specific preparation steps depend on the type of asset being deployed) and invokes the deployer service on the Mediator. The prepared asset is submitted as input to this service.
3	The deployer service deploys the asset in the Mediator.
4	If the deployment is successful, the deployer service returns a success message and data that is pertinent to the deployed asset. If the deployment is unsuccessful, the deployer service returns a failure message.
5	The Deployment Manager on CentraSite logs information about the deployment in the Deployment log. If the deployer service returned specific data about the asset, the asset's metadata is updated as needed in the registry/repository.

For more information, see ["What Happens When You Deploy a Virtualized Service?"](#) on page 154.

Who Can Deploy Virtualized Services to Targets?

To deploy virtualized services to targets, you must belong to a role with the following permissions:

- Create Assets —OR— Manage Assets
- Manage Runtime Policies (required to configure virtualized services)
- Manage Lifecycle Models (required to change state of virtualized services)
- Mediator Publisher (required to deploy virtualized services)

For more information about roles and permissions, see *Getting Started with CentraSite*.

Note: Mediator exposes several Web service operations to allow CentraSite to manage deployed assets. This Web service is invoked by CentraSite any time a user deploys or undeploys a virtual service to Mediator. A Mediator's configuration details page includes the `User Name` and `Password` fields which identify an Integration Server user who is permitted to execute the Integration Server services associated with Mediator's deployer service. After installation, only members of the Integration Server "Administrators" group are allowed to invoke these services. However, administrators have the flexibility to allow their own users or groups to invoke them. Access to these services is controlled by an ACL, called `MediatorDeployer`. Initially, only the predefined "Administrators" group is assigned to this ACL. An Integration Server administrator can remove this group and add other groups or individual users. For example, you can create your own deployer group, (for example, "MyDeployers") and add Integration Server user IDs to this group. Then, the user must update the `MediatorDeployer` ACL by removing the "Administrators" group and adding the "MyDeployers" group. Now, on the target's configuration detail page, you can specify any user ID that belongs to "MyDeployers" group.

Conditions that Must be Satisfied for Effective Deployment of Virtualized Services

To deploy a virtualized service to a Mediator, the following conditions must be met:

- Ensure that you have the Mediator Publisher role. Only users with this permission can deploy a virtual service. CentraSite will not enable the deployment controls for any other users. For more information, see ["Who Can Deploy Virtualized Services to Targets?" on page 153](#).
- Ensure that the run-time policies for the virtualized service are active. This is indicated in the **Policies** profile on the virtualized service's detail page. If a policy is inactive, you must activate it as described in ["Activating a Run-Time Policy" on page 135](#).

- Ensure that the gateway to which the virtual service will be deployed has already been created, as described in <placeholder>.
- Ensure that the gateway's specified deployment URL is active and the user credentials of Integration Server are valid. To check this, go to the gateway's detail page and click the **Publish** button. If the connection is not active and valid, activate the deployment endpoint and modify the user credentials as required.
- If the virtualized service is under the control of an active lifecycle model (LCM), ensure that:
 - The virtualized service is in a "deployable" lifecycle state. If you are not certain of what the "deployable" lifecycle state is, consult your CentraSite administrator.
 - The virtualized service has a design-time policy that includes the Change Deployment Status action and it is set to Yes. This action specifies whether the service is eligible for deployment. For more information, see the description of the action in the *CentraSite Developer's Guide*.

If these conditions are not satisfied, all or part of the deployment user interface controls will be disabled when you view the virtual service.

What Happens When You Deploy a Virtualized Service?

When you deploy a virtualized service to one or more Mediator gateways, keep the following in mind.

- **CentraSite automatically validates the service's run-time policy (or policies)** to ensure that:
 - Any action that appears in a single policy multiple times is allowed to appear multiple times.
 - All action dependencies are properly met.

CentraSite will inform you of any violation, and you will need to correct the violations before deploying the service. For more information about dependencies and which actions can be included multiple times in a single policy, see "[Built-In Run-Time Actions Reference for Virtual Services](#)" on page 347.

- **You must make modifications to deployed assets in CentraSite.**

If you need to modify a virtualized service that is already deployed, you must modify it in CentraSite and then redeploy it to Mediator. Mediator does not monitor CentraSite for updates to deployed assets. If you make changes to a virtual service's processing steps, for example, you must manually redeploy the virtual service to put those changes into effect.

- **You cannot make changes to a run-time policy while it is active.**

To make changes to a policy after it has been switched to the active state you must do one of the following:

- Switch the policy to the Suspended state (to deactivate it), update the policy and then switch it back to the Productive state (to reactivate it).

OR

- Create a new version of the policy, make your changes to the new version of the policy and then switch the new version to the Productive state. Switching the new version of the policy to the Productive state will automatically Retire (and deactivate) the old version.

If you need to update a run-time policy that is already deployed with virtual services that are in production, always use the second method described above (i.e., create a new version of the policy). If you use the first method, which requires you to suspend the existing policy, your production services will be running without the policy while you are making your revisions to it.

- **When you deploy a virtualized service, CentraSite generates a VSD.**

When you deploy a virtualized service to a Mediator, CentraSite generates an XML document called a virtual service definition (VSD). The VSD defines the virtualized service for Mediator, and contains all the run-time policies and resources required to deploy the virtualized service to Mediator.

- **You should not manually edit the endpoint information for virtualized services.**

CentraSite automatically updates the service's CentraSite endpoint to its Mediator endpoint. You can view the Mediator endpoint on the virtualized service's detail page in CentraSite. Because the endpoint information for virtualized services is generated and updated by CentraSite, unlike when managing native services, you should not manually add endpoints to a virtualized service. Instead, allow CentraSite to generate and manage the endpoints for the virtualized services that you deploy.

However, you can deploy multiple virtualized services for a single native service to make the service available over multiple transports and/or security mechanisms. For details about managing endpoints, see *Getting Started with CentraSite*.

- **If deployment fails, the status is set to "Failed" and the failure is logged.**

If Mediator encounters a problem deploying or redeploying a virtualized service, it sets the service's Deployment Status to "Failed" and sends a message to CentraSite describing the problem. This failure is also logged to Mediator. In this case, it is up to the CentraSite or Mediator administrator to take corrective action and redeploy the service manually from CentraSite.

If the reason for the failure is that the Mediator instance is unavailable, and then you restart the Mediator instance, it loads all information about any previously deployed assets.

Deploying, Undeploying, and Redeploying Virtualized Services

You can deploy, undeploy and redeploy virtualized services (of any type) to a Mediator in any of the following ways:

- **From the virtualized service's detail page**

You can deploy, undeploy and redeploy a virtualized service to one or more Mediator (see ["Deploying Virtualized Services from a Service's Detail Page"](#) on page 156).

■ From the Operations > Deployment page

You can deploy, undeploy and redeploy *multiple* virtualized services to a Mediator in a single step (see ["Deploying Virtualized Services from the Operations > Deployment Page"](#) on page 158).

■ Using run-time commands

You can use the CentraSite Command facility to deploy, undeploy and redeploy virtualized services to a Mediator individually or in bulk (see ["Deploying Virtualized Services Using Command Line Tool"](#) on page 164).

■ Running a batch process

You can run a batch process to deploy, undeploy and redeploy multiple virtualized services to a Mediator in a single step (see ["Deploying Virtualized Services Using a Batch Process"](#) on page 167).

If You Migrate Virtualized Services from a Pre-8.2 Release

If you have virtualized services that were created prior to version 8.2, those virtualized services will continue to hold the deployment metadata generated by CentraSite 8.0. Although this metadata is not applicable in CentraSite 8.2, and will not affect deployment in 8.2, we strongly recommend that you perform the following steps:

1. Undeploy all virtualized services from CentraSite 8.0.
2. Upgrade to CentraSite 8.2.
3. Ensure that all gateway endpoints are configured correctly.
4. Deploy all virtualized services from CentraSite 8.2.

Note: Please be aware that the new synchronous deployment model does not support subscriptions and subscription services.

Deploying Virtualized Services from a Service's Detail Page

The following procedure describes how to deploy, undeploy, and redeploy a single virtualized service to one or more Mediator gateways, using the service's **Deployment** profile.

To deploy, undeploy or redeploy a virtualized service

1. In CentraSite Control, display the details page of virtualized service. For procedures for this step, see ["Viewing or Editing Virtualized Services"](#) on page 120.
2. Select the virtualized service's **Policies** profile and ensure that all of the service's run-time policies are Active. If not, activate them as described in ["Activating a Run-Time Policy"](#) on page 135.

3. Ensure that the virtualized service has a design-time policy that includes the Change Deployment Status action and it is set to Yes. This action specifies whether the service is eligible for deployment. For more information, see the description of this action in the *CentraSite Developer's Guide*.
4. Make sure you switch the service to an active, ready-to-deploy state, as follows. (If you do not know which state to select, you will need to examine your organization's lifecycle model for Service objects or consult an administrator.)
 - a. In the **Actions** menu, select the **Change Lifecycle State** option.
 - b. Select the lifecycle state to which you want to switch the asset and click **OK**. (The list will contain only the states that you are permitted to assign to the service.)

If the state change requires approval, CentraSite will initiate an approval workflow and your request for a state change will be submitted to the appropriate approvers. While the request is awaiting approval, the service will appear in a pending state.
5. After the lifecycle state has been successfully changed, select the virtualized service's **Deployment** profile.

The **Deployment** profile will display the following information.

Column	Description
Target	The target on which the service is deployed.
Target Type	The type of target on which the service is deployed (for example, webMethods Integration Server or webMethods Insight).
Description	Description of the target.
Deployment Status	The deployment status of the service (for example, Deployed or Failed).
Date Deployed	The date/time that the deployment occurred.

6. Click the **Deploy** button, select the Mediator gateway(s) to which you want to deploy the virtualized service, and click **OK**.
7. The deployment process is carried out by a synchronous mechanism between the CentraSite and the Mediator:
 - a. CentraSite pushes the virtualized service that is ready for deployment to the Mediator.

- b. Instantly, the Mediator deploys the virtualized service that was received from CentraSite (along with its effective run-time policy), and notifies CentraSite when the deployment process is complete.

For more information, see ["What Happens When You Deploy a Virtualized Service?" on page 154](#).

Important: If the status shown in the **Deployment Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to deploy the virtualized service successfully. If the deployment process failed, identify and correct the error and then try deploying the virtualized service again.

8. To undeploy the virtualized services, select the services' check boxes, and choose **Undeploy** from the Actions menu.

If the undeployment is successful, Mediator's deployer service returns a success message, and data that is pertinent to the undeployed virtual service. In addition, CentraSite's Deployment Manager logs information about the undeployment in the Deployment log. If the undeployment is unsuccessful, the deployer service returns a failure message.

Important: If the status shown in the **Deployment Status** column does not automatically switch to **Undeployed**, click the **Refresh** button to determine whether CentraSite was able to undeploy the virtualized services successfully. If the undeployment process failed, identify and correct the error and then try undeploying the virtualized services again.

9. To redeploy the virtualized service, select the services' check boxes, and choose **Redeploy** from the Actions menu.

Important: If the status shown in the **Deployment Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to redeploy the virtualized services successfully. If the redeployment process failed, identify and correct the error and then try redeploying the virtualized services again.

10. Examine the deployment log that is displayed by CentraSite Control and check for any errors that occurred during the deployment process.

Deploying Virtualized Services from the Operations > Deployment Page

The following procedure describes how to deploy, undeploy and redeploy multiple virtualized services to a Mediator in a single step.

To deploy, undeploy or redeploy a virtualized service

1. In CentraSite Control, go to **Operations > Deployment**.
2. On the **Deployed Assets** tab, click **Deploy**.

3. In the **Select a Target and Services to be Deployed on the Selected Target** dialog box, perform a keyword or advanced search to display the virtualized services that are ready for deployment.
 - If you want to perform a keyword search and you need procedures for this step, see ["Using a Keyword Search to Select a Target or Service to Deploy"](#) on page 161.
 - If you want to perform an advanced search and you need procedures for this step, see ["Using a Advanced Search to Select a Target or Service to Deploy"](#) on page 163.
4. Click **OK**.
5. The deployment process is carried out by a synchronous mechanism between the CentraSite and the Mediator:
 - a. CentraSite pushes the virtualized service that is ready for deployment to the Mediator.
 - b. Instantly, the Mediator deploys the virtualized service that was received from CentraSite (along with its effective run-time policy), and notifies CentraSite when the deployment process is complete.

For more information, see ["What Happens When You Deploy a Virtualized Service?"](#) on page 154.

The **Deployed Assets** tab will display the following information.

Column	Description						
Pending Changes	Icons indicating the deployment status of the virtualized services.						
	<table border="1"> <thead> <tr> <th>Icon</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td></td> <td>Service is deployed to the target.</td> </tr> <tr> <td></td> <td>Service is pending deployment to the target.</td> </tr> </tbody> </table>	Icon	Description		Service is deployed to the target.		Service is pending deployment to the target.
Icon	Description						
	Service is deployed to the target.						
	Service is pending deployment to the target.						
Deployment ID	The deployment ID of the virtualized service.						
Name	Name of the virtualized service.						
Status	The deployment status of the virtualized service (e.g., Deployed or Failed).						
Date/Time	The date/time that the deployment occurred.						

Column	Description
User ID	The user-ID of the Integration Server user to be used for the deployment operation.
Type	Modification that is been performed on the deployed service.
Label	Description
Processing Step Changes	Reflects any change that is performed in the virtualized service's Processing Steps profile. For example, modifying the HTTP authentication mode.
Runtime Policy Changes	Reflects any change that is performed in the runtime policy associated to the virtualized service. For example, deactivating an associated runtime policy.
WSDL Changes	Reflects any change that is performed in the virtualized service's asset file. For example, modifying an existing asset file or uploading a new asset file.

Important: If the status shown in the **Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to deploy the virtualized services successfully. If the deployment process failed, identify and correct the error and then try deploying the virtualized services again.

- To undeploy the virtualized services, select the services' check boxes, and choose **Undeploy** from the Actions menu.

Important: If the status shown in the **Status** column does not automatically switch to **Undeployed**, click the **Refresh** button to determine whether CentraSite was able to undeploy the virtualized services successfully. If the undeployment process failed, identify and correct the error and then try undeploying the virtualized services again.

- To redeploy the virtualized service, select the services' check boxes, and choose **Redeploy** from the Actions menu.

Important: If the status shown in the **Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to redeploy the virtualized services successfully. If the redeployment

process failed, identify and correct the error and then try redeploying the virtualized services again.

8. Examine the deployment log that is displayed by CentraSite Control and check for any errors that occurred during the deployment process.

Selecting a Target and Services to be Deployed on the Selected Target

You can use CentraSite's Search feature to select target and services to be deployed on the selected target. You can perform a keyword search or an advanced search. For more information, see ["Using a Keyword Search to Select a Target or Service to Deploy" on page 161](#) and ["Using a Advanced Search to Select a Target or Service to Deploy" on page 163](#).

Using a Keyword Search to Select a Target or Service to Deploy

The keyword search is an easy to use search facility in which you can specify arbitrary search patterns.

You can search for all virtualized services that contain one or more specified keywords (i.e., text strings) in the virtualized service's string attributes (virtualized service name, description, and so on.).

Conventions for Keyword Searches

The conventions for keyword searches are as follows:

- A keyword search consists of 1-n search keywords. Multiple keywords are space separated. If multiple keywords are given, a logical disjunction (OR) is implied.
- A keyword is treated as partial text which can occur at the beginning of the searched strings. The `starts with` semantics are implied.

Example: If the keyword is `AustralianPostCode`, then the following matches are returned: A sample VS for `AustralianPostCode` as well as `AustralianPostCodeVService`.
- If quotes (" ") exist around a phrase, then a search is performed on the exact phrase within the quotes. A space within a quoted phrase is considered as a space character and not as a logical operation. To force the keyword search to treat the quote characters as a normal character, precede the quote character with a backslash (\). If you want to include the backslash character itself in the search, type two backslashes.
- You can mix and match any number of words and quoted phrases within the keyword field.
- The search is neither case nor accent sensitive, even within a quoted phrase.
Example: A search for `AustralianPostCode` will return the same results as a search for `AUSTRALIANPOSTCODE` or `Australianpostcode`.
- If you enter a string that contains an odd number of double-quote characters, then the last double-quote character is ignored when the search is performed.

- If the keyword search input field is empty when the search is executed, the search returns all available virtualized services.
- The keyword search can include wildcard characters.

Wildcard Characters

The available wildcard characters are:

Character	Usage
* or %	If you use the percent symbol (%) or the asterisk (*), CentraSite replaces the wildcard symbol with as many characters as necessary to find a match. For example, an entry of A%n returns both Amazon and American. If you enter *al, then CalcService, Calendar and AustralianPostCode all fit the search criteria.
? or _	If you use the question mark (?) or the underscore (_), CentraSite replaces the wildcard symbol with a single character in order to find a match. Example: AustralianPostCode?VService matches any character for ?.

You can use a wildcard character at any point in the keyword text, and multiple times throughout the keyword text. If you enter a wildcard character in the middle of a string, for example `cat*dog`, then at least one of the searched attributes must contain the string in order for the asset or supporting document to be included in the result set.

If a wildcard character between two words is surrounded by spaces, such as `word1 * word2`, the wildcard will match one word.

Note: Keep the following in mind:

- Certain non-alphanumeric characters that can appear in the name of a virtualized service are currently ignored by CentraSite's wildcard mechanism when you include them in a keyword search. In particular, the hyphen (-) is ignored. Thus, if you have created the virtualized services `AustralianPostCodeVService-1` and `AustralianPostCodeVService_1`, the wildcard search for `AustralianPostCodeVService?` will find `AustralianPostCodeVService_1` but not `AustralianPostCodeVService-1`.
- The percent (%) character acts as a word delimiter when it appears in the text to be searched. Thus, for example, if the description field of a virtualized service contains the text `abc%def` (the characters a, b, c, %, d, e, f), this is treated by the search mechanism as two adjacent words `abc` and `def`. A wildcard search such as `abc*def` looks for a single word beginning with `abc` and ending with `def`, so the search will not find this asset.

Performing a Keyword Search

To search by keyword

1. In CentraSite Control, go to **Operations > Deployment**.
2. On the **Deployed Assets** tab, click **Deploy**. This opens up the **Select a Target and Services to be Deployed on the Selected Target** dialog.
3. In the text box, type the keyword(s) to search for. You can use one or more wildcards to specify the keywords.

If you leave the text box blank, or enter just a wildcard, the entire set of virtualized services is returned.

CentraSite returns the virtualized services that match the search criteria. The search looks for the keyword(s) in the virtualized service's name, type and description attributes.

Using a Advanced Search to Select a Target or Service to Deploy

CentraSite's advanced search capabilities allow you to search for virtualized services on the basis of service types and targets.

To search using service type and target

1. In CentraSite Control, go to **Operations > Deployment**.
2. On the **Deployed Assets** tab, click **Deploy**.
3. In the **Select a Target and Services to be Deployed on the Selected Target** dialog box, do the following:
 - a. In the field **Browse By**, select a virtualized service type. As a result, only virtualized services that are classified with this service type will be displayed.
 - If you do not specify a virtualized service type in the field **Browse by**, CentraSite Control displays a list of all virtualized services belonging to the CentraSite.
 - If you specify a virtualized service type in the field **Browse by**, CentraSite Control displays a list of all virtualized services that belong to the specified service type.

There are several generic entries in the drop-down list for the **Browse by** field. These are:

- **[All]**

This lists all virtualized services that are available in a deployable state.
- **[Virtual Service]**

This lists all virtual Web services that are available in a deployable state.
- **[Virtual REST Service]**

This lists all virtual REST services that are available in a deployable state.

- **[Virtual XML Service]**

This lists all virtual XML services that are available in a deployable state.

- b. In the field **Target**, select a target for deploying the selected services.
- c. Select the virtualized services that you want to deploy on the selected target.

4. Click **OK**.

Deploying Virtualized Services Using Command Line Tool

You can perform the following deployment tasks by executing commands in the command line interface `CentraSiteCommand.cmd` (Windows) or `CentraSiteCommand.sh` (UNIX) of CentraSite. The tool is located in `<CentraSiteInstallDir>/utilities`.

- Deploy or undeploy virtual services to Mediator.
- Perform a “bulk deploy”, a “bulk undeploy”, a “bulk redeploy”, and a “bulk clean redeploy”.

If you start the command line tool with no parameters, you receive a help text summarizing the required input parameters.

The parameters of the command are case-sensitive, so for example the parameter `-url` must be specified as shown and not as `-URL`.

Deploy a Virtualized Service to Mediator

To deploy a virtualized service to Mediator, use a `deploy` command of the following format:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd deploy [-url
<CENTRASITE-URL>] -user <USER-ID> -password <PASSWORD> -virtualService
<VIRTUAL-SERVICE> -gateway <GATEWAY>
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the `deploy` command:

Input Parameter	Description
<code>-url</code>	The fully qualified URL (<code>http://localhost:53307/CentraSite/CentraSite</code>) for the CentraSite registry/repository.
<code>-user</code>	The user ID of a user who has the CentraSite Administrator role.

Input Parameter	Description
-password	The password of the user identified by the parameter -user.
-virtualService	The name or key of the virtual service.
-gateway	The gateway to which the virtual service identified by the parameter -virtualService is to be deployed.

For example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd deploy -url
http://localhost:53307/CentraSite/CentraSite -user Administrator -
password manage -virtualService VS1 -gateway Gateway1
```

Undeploy a Virtualized Service from Mediator

To undeploy a virtualized service from Mediator, use a undeploy command of the following format:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd undeploy [-url
<CENTRASITE-URL>] -user <USER-ID> -password <PASSWORD> -virtualService
<VIRTUAL-SERVICE> -gateway <GATEWAY>
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the undeploy command:

Input Parameter	Description
-url	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
-user	The user ID of a user who has the CentraSite Administrator role.
-password	The password of the user identified by the parameter -user.
-virtualService	The name or key of the virtual service.
-gateway	The gateway from which the virtual service identified by the parameter -virtualService is to be undeployed.

For example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd undeploy -
url http://localhost:53307/CentraSite/CentraSite -user Administrator -
password manage -virtualService VS1 -gateway Gateway1
```

Deploy, Undeploy or Redeploy Multiple Virtualized Services in Mediator

To deploy, undeploy or redeploy multiple virtualized services in Mediator, use commands of the following formats:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd bulk deploy [<parameters>]
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd bulk undeploy [<parameters>]
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd bulk redeploy [<parameters>]
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd bulk clean redeploy [<parameters>]

<parameters> :
[-url <CENTRASITE-URL>]
-user <USER-ID>
-password <PASSWORD>
-gateway <GATEWAY>]
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the above commands:

Input Parameter	Description
-url	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
-user	The user ID of a user who has the CentraSite Administrator role.
-password	The password of the user identified by the parameter -user.
-gateway	The gateway to/from which the virtual services are to be deployed/undeployed/redeployed.

For example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd bulk deploy -
url http://localhost:53307/CentraSite/CentraSite -user Administrator -
password manage -gateway Gateway1
```

Configuring API Key Header Using Command Line Tool

The command line tool in CentraSite is used to specify the API key header name. This API key header name specified is sent to Mediator as a part of publishing gateways

and is used by Mediator for authentication of the API at runtime. Once the Mediator gateway is published, the new header name is set in Mediator .

To set the API key header name, use the following command:

```
set APIKey header name -url <CENTRASITE-URL> -user <USER-ID> -password <PASSWORD>
-apiKeyHeaderName <API-KEY-HEADER-NAME
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the above command:

Input Parameter	Description
-url	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
-user	The user ID of the registered CentraSite user. For example, a user who has the CentraSite Administrator role.
-password	The password for the registered CentraSite user identified by the parameter <code>-user</code> .
-apiKeyHeaderName	The custom API key header used to send API key to Mediator The default API key header is <i>x-CentraSite-APIKey</i> .

Note: If a custom header for the **API key header** property is not defined, the *x-CentraSite-APIKey* is used as the default value. To retrieve the current API key header, the `get APIKey header name` command is used.

Deploying Virtualized Services Using a Batch Process

Use `Runtime.deployment.Deployer` when you do not have access to a browser or graphical user interface environment, and you want to perform deployment tasks. You can also use `Runtime.deployment.Deployer` when you want to automate deployment tasks through batch processes.

An automated deployment through a batch mode can be initiated by configuring the `DeploymentConfiguration.properties` file located in the URL `http://<host>:53307/CentraSite/CentraSite/ino:dav/ino:dav/projects/CentraSite/configuration`.

Specifying a Deploy Batch Size

BatchSize is the maximum number of virtualized services to be pushed to the Mediator before a syncpoint is taken. The default BatchSize is 50. To improve performance, you can set a BatchSize to define the maximum number of virtualized services to be pushed between two syncpoints using the property line:

```
com.softwareag.centrasite.runtime.deployment.DeployBatchSize=50
```

The BatchSize property can be set at any time. If a bulk deployment is already in progress, the current batch is sized according to the previous batch size. Subsequent batches use the new size. Suppose if the BatchSize is set to zero and changed while a deploy operation is already in progress, that operation loads the data as a single batch. Any subsequent deploy operations on the same CentraSite Control use the new BatchSize.

Specifying a Transaction Timeout

TransactionTimeout specifies the maximum time, in milliseconds, allowed for deployment operations (deployment or undeployment) that were pushed to Mediator to respond. Any such operations that do not respond before this timeout occurs are rolled back. The default TransactionTimeout is 6000 (ms). To improve performance, you can set a TransactionTimeout to define the maximum time for the deployment operations to respond using the property line:

```
com.softwareag.centrasite.runtime.deployment.TransactionTimeout=60000
```

For example, if a deployment operation attempts to set a transaction timeout of 360 seconds, and the TransactionTimeout setting is 300 seconds, the TransactionTimeout setting of 300 seconds is used. After the TransactionTimeout of 300 seconds the deployment operations roll back.

Note: If set to 0, the transaction will not time out.

Viewing the Deployment History Log

The Deployment History Log contains information about the virtualized services that CentraSite has pushed to the Mediator for deployment.

To view the Deployment History Log, you must belong to the Mediator Publisher role. To see the list of predefined roles that include this permission, see the *CentraSite Administrator's Guide*.

The following procedure describes how to view the deployment history log. To view this log, you must belong to a Mediator Publisher role.

- If you belong to one of the following roles, you can view all entries in the deployment history log.
 - CentraSite Administrator
 - Organization Administrator

- Operations Administrator
- If you belong to the Organization Administrator role for an organization, you can view the deployment history log entries for virtualized services that were created by users in your organization.
- If you do not belong to either of these roles, but you have the Mediator Publisher role, you can view the deployment history log entries for virtualized services that you created.

To view the Deployment History Log

1. In CentraSite Control, go to **Operations > Deployment**.
2. Click the **Deployment History** tab to view the list of all virtualized services that are sent to the Mediator.
3. If you want to filter the list, type a partial string in the **Search** field. CentraSite applies the filter to the **Name** column. The **Search** field is a type-ahead field, so as soon as you enter any characters, the display will be updated to show only those virtualized services whose name contains the specified characters. The wildcard character “%” is supported.

The **Deployment History** tab provides the following information about each virtualized service.

Column	Description
Rule ID	The synchronization rule-ID that CentraSite automatically generates up on creation of the Mediator in CentraSite Control.
Name	The name assigned to the virtualized service.
Type	The type of the virtualized service (say, Web Service, XML Service or REST Service).
Version	The user-assigned version identifier for the virtualized service.
Target Name	The name of the target on which the virtualized service is deployed.
Action	The deployment action of the virtualized service on the Mediator (for example, Deployed, Undeployed).
Status	The deployment status of the virtualized service on the Mediator (for example, Success, Failed).

Column	Description
Date	The date/time that the virtualized service has deployed, redeployed or undeployed.

- To view details of a particular deployment workflow, click the hyperlinked value in the **Name** column.

Deleting a Deployment Activity Log

When you delete an activity log, keep the following points in mind:

- To delete an activity log, you must belong to one of the following roles:
 - CentraSite Administrator
 - Organization Administrator
 - Operations Administrator
- Remember deleting an activity log via the **Deployment History** tab does not affect the deployment status of a virtualized service. However, deleting an activity log through the **Deployed Assets** tab will undeploy a virtualized service from the Mediator and the status automatically switches to **Undeployed**.
- The **Deployment History** tab contains log entries for every deployment operation (such as Deployed, Undeployed and so on) of a virtualized service. However, the **Deployed Assets** tab contains log entry only for a Deployed operation of the virtualized service. Once you undeploy a virtualized service, the log entry will automatically get removed from the **Deployed Assets** tab.
- Be aware that you can never delete the last Deployed activity log of a virtualized service using the **Deployment History** tab.

You can delete a deployment activity log in any of the following ways:

- Using the **Deployed Assets** tab on the Deployment page as described in "[Performing a Delete Operation Using the Deployed Assets Tab](#)" on page 170. This procedure enables you to delete the activity logs of virtualized services in the CentraSite Control.
- Using the **Deployment History** tab on the Deployment page as described in "[Performing a Delete Operation Using the Deployment History Tab](#)" on page 171. This procedure enables you to delete the activity logs of virtualized services in the CentraSite Control.

Performing a Delete Operation Using the Deployed Assets Tab

You delete the “deployed” and “redeployed” activity logs of virtualized services via the **Deployed Assets** tab.

You can delete multiple activity logs in a single step.

To delete one or more activity logs

1. In CentraSite Control, go to **Operations -> Deployment**.
2. On the **Deployed Assets** tab, mark the checkbox next to the name of each activity log that you want to delete. (You can select multiple logs.)
3. In the **Actions** menu, click **Delete**.

When you are prompted to confirm the delete operation, click **OK**.

Each selected log is permanently removed from the CentraSite registry/repository. The activity logs in the virtualized service's **Deployment** profile and target's **Service** profile are not affected.

Performing a Delete Operation Using the Deployment History Tab

You delete the “deployed”, “undeployed”, “redeployed” and “failed” activity logs of virtualized services using the **Deployment History** tab.

You can delete multiple activity logs in a single step.

To delete one or more activity logs

1. In CentraSite Control, go to **Operations -> Deployment**.
2. On the **Deployment History** tab, mark the checkbox next to the name of each activity log that you want to delete. (You can select multiple logs.)
3. Choose **Delete**.

When you are prompted to confirm the delete operation, click **OK**.

Each selected log is permanently removed from the CentraSite registry/repository. The activity logs in the asset's **Deployment** profile and target's **Service** profile are not affected.

Securing Communications with for Synchronous Deployment

An administrator can configure CentraSite to use either of the following kinds of authentication:

- HTTP Basic authentication
- HTTPS (Secure Sockets Layer (SSL)) authentication

Configuring CentraSite to use SSL authentication provides secure communications for the deployment.

This section explains how SSL works with CentraSite (which acts as the client) and Mediator (which acts as the server). This section also provides the information you need to configure both the client and server sides for SSL authentication.

Anatomy of a SSL Connection

It is useful to conceptualize a CentraSite SSL connection in terms of a SSL client and a SSL server. The request for an SSL connection originates from a client.

During the SSL handshake process, the Mediator acting as the SSL server responds to the request for a connection by presenting its SSL credentials (an X.509 certificate) to the requesting CentraSite client. If those credentials are authenticated by the CentraSite client, either:

- An SSL connection is established and information can be exchanged between the CentraSite and Mediator.
- OR—
- The next phase of the authentication process occurs, and the Mediator requests the SSL credentials of the CentraSite. If the Mediator verifies those credentials (that is, the client's identity), an SSL connection is established and information exchange can take place.

SSL Connection Type

The types of SSL connection referred to above are termed *one-way* and *two-way* SSL authentication:

- In a *one-way* SSL connection, client authenticates the credentials of server in preparation for setting up a secure transaction. In most cases, the server knows nothing about the client's identity because verification of its credentials is not required. When desired, however the client can be authenticated by means such as basic username/password.

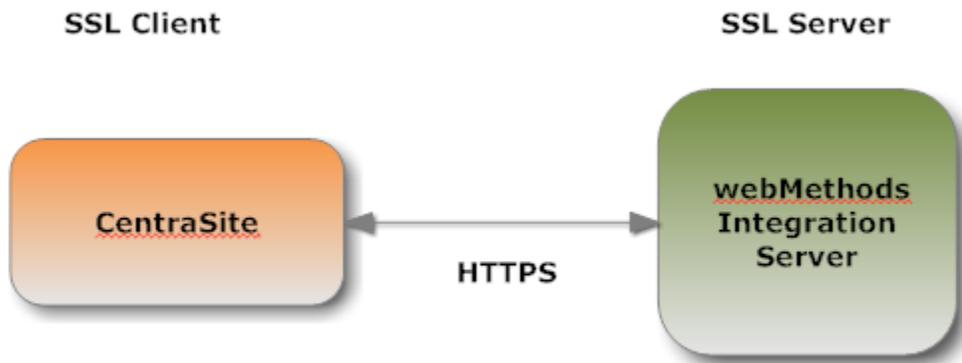
This type of connection is typically one where CentraSite needs to verify the authenticity of the Mediator without itself needing to be authenticated. As a result, configurations on the CentraSite side are not actually required for this one-way connection.

- In a *two-way* SSL connection, both client and server must authenticate each other's credentials before an SSL connection is established and information can be exchanged.

Unlike a one-way SSL connection, both CentraSite and the Mediator require access to each other's SSL certificates in order to authenticate each other, establish an SSL connection, and transmit information. Compared to a one-way connection, a two-way SSL connection provides a much higher level of security.

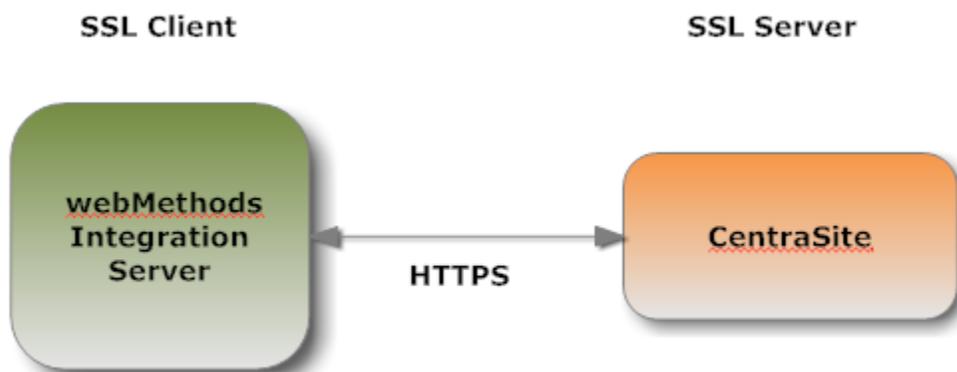
As an SSL Client

When the CentraSite submits a HTTPS request to the Mediator, CentraSite is the SSL client and the Mediator with which it is communicating acts as the SSL server.



As an SSL Server

When the Mediator submits a request to CentraSite via HTTPS, and a two-way SSL connection is established, the Mediator acts as the SSL client and the CentraSite acts as the SSL server.



Roadmap for Configuring SSL

The following table provides a high-level roadmap for configuring SSL on CentraSite.

Task	Activities	Notes
Create CentraSite keys and certificates	<ul style="list-style-type: none"> ■ Generate a public key/private key pair. ■ Generate a certificate signing request (CSR) and send to the certificate authority (CA) for signing. 	<p>Required for one-way and two-way SSL connections.</p> <p>Refer to the documentation for Java <i>keytool</i> or</p>

Task	Activities	Notes
	<ul style="list-style-type: none"> Receive validated certificate from the CA. Import signed certificate into a keystore. 	your certificate management tool.
Create keystore and truststore for CentraSite	<ul style="list-style-type: none"> Create a keystore and import the signed certificate and private key. Create a truststore and import the certificate of the signing CA. Store the keystore and truststore in a secure CentraSite certificates directory. <div style="border: 1px solid gray; background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Important: You use a Java <i>keytool</i> to create the keystore, you cannot import an existing private key. You can use other tools such as OpenSSL or Portecle.</p> </div>	<p>Required for one-way and two-way SSL connections.</p> <p>Refer to the documentation for your certificate management tool.</p>
Obtain certificates of webMethods Mediator	<p>Use the CentraSite truststore to save:</p> <ul style="list-style-type: none"> Signed certificate of the Mediator. Signed certificate of the CA for the Mediator's SSL certificate. 	Required for one-way and two-way SSL connections.

Creating Keys and Certificates

Use a standard certificate management tool, such as OpenSSL or Portecle, to generate a private/public key pair for CentraSite. Then, place the public key in a certificate signing request (CSR).

After creating the CSR, send to the CA to sign the CSR. Request the certificate in DER format. If you receive a certificate in PEM format (or any format other than DER), you need to convert it to DER format.

The signing CA's certificate attests to the identity of the CA that signed the digital certificate for the CentraSite. The CA should send this certificate to you when it sends you the digital certificate for the CentraSite.

Once you receive your signed certificate from the CA, you need to import the certificate into a keystore. You will then have an SSL certificate and private key to use with CentraSite.

Note: The above process is described in general terms. The procedures may vary somewhat, depending upon the CA that you use.

Creating a Keystore and Truststore

Keystores and truststores are files that function as repositories for storage of keys and certificates necessary for SSL authentication, encryption/decryption, and digital signing/verification services. Keystores and truststores provide added layers of security and ease of administration, compared to maintaining the keys and certificates in separate files.

For information about creating keystores and truststores, importing keys and certificates into keystores and truststores, and other operations with these files, refer to the documentation for your certificate management tool.

For information about using CentraSite with keystores and truststores, see "[Keystores and Truststores](#)" on page 175.

Obtaining the Certificates and Keys of the webMethods Mediator

If your CentraSite will submit HTTPS requests to the Mediator, the CentraSite will be acting as a client and will receive certificates from this Mediator. In order for these transactions to work, your CentraSite must have copies of their public keys and signing CA certificates. For information on importing Mediator certificates to CentraSite and setting up client authentication, see *webMethods Integration Server Administrator's Guide*

Keystores and Truststores

CentraSite stores its private keys and SSL certificates in keystore files and the trusted roots for the certificates in truststore files. Keystores and truststores are secure files with industry-standard file formats.

Keystore File

CentraSite uses a special file called a *keystore* to store SSL certificates and keys.

A keystore file contains one or more pairs of a private key and signed certificate for its corresponding public key. The keystore should be strongly protected with a password, and stored (either on the file system or elsewhere) so that it is accessible only to administrators.

Keystore File Formats

The default, certificate file format for a CentraSite keystore is JKS (Java keystore). Java keystore is a commonly used, standardized, certificate file format that provides a high degree of portability. PKCS#12 is another format you can use for a keystore. Other keystore types can be made available by:

- Loading additional security providers
- Setting the `watt.security.keyStore.supportedTypes` property.

HSM-Based Keystores

Under certain conditions, Mediator supports the use of keystore files stored on a Hardware Security Module (HSM). Integration Server supports HSM-based keystores

for ports, but not for other components. You cannot use HSM-based keystores with remote server aliases, outbound certificates, an internal server port, WS-Security, or Integration Server public services.

Creating a Keystore

You can create and manage CentraSite keystores at the command line using `keytool`, a Java certificate editor.

You can also use other standard tools such as OpenSSL and Portecle.

Note: Software AG does not provide its own set of keystore utilities for creating or managing keystore files.

Truststore File

CentraSite uses a *truststore* to store its trusted root certificates, which are the public keys for the signing CAs. Although a truststore can contain the trusted roots for entire certificate chains, there is no requirement for the organization of certificates within a CentraSite truststore. It simply functions as a database containing all the public keys for CAs within a specified trusted directory.

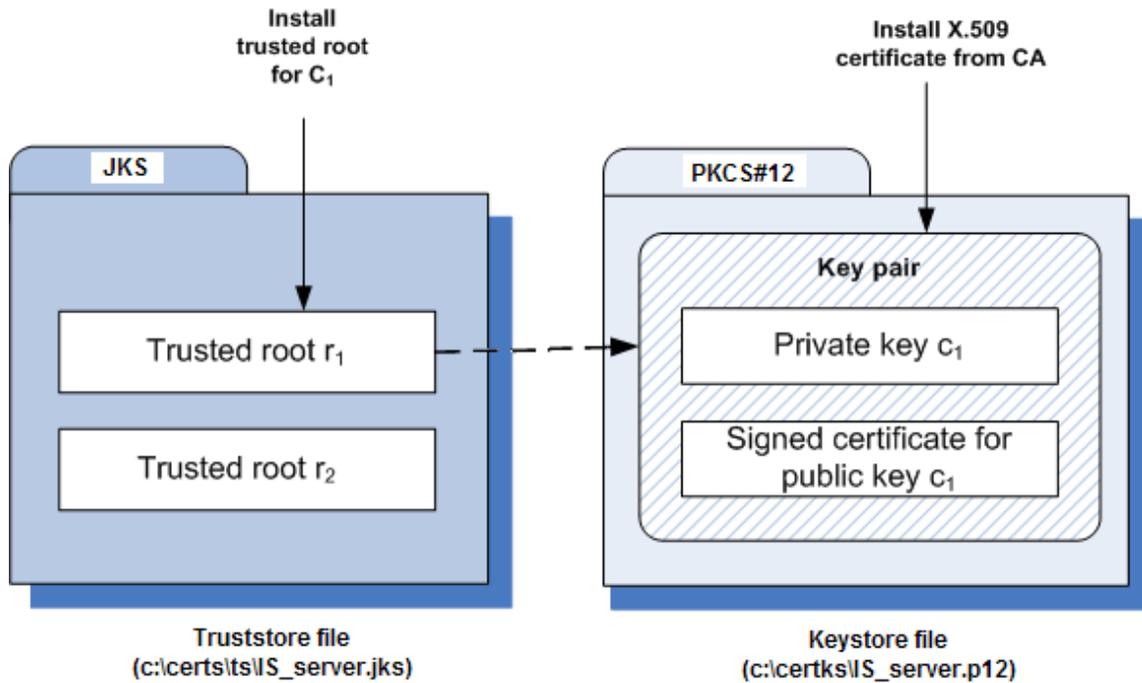
Truststore File Formats

CentraSite uses a *truststore* to store its trusted root certificates, which are the public keys for the signing CAs. Although a truststore can contain the trusted roots for entire certificate chains, there is no requirement for the organization of certificates within a CentraSite truststore. It simply functions as a database containing all the public keys for CAs within a specified trusted directory.

How Uses a Keystore and Truststore

For a CentraSite service to be SSL authenticated, it must have a valid, authorized X.509 certificate installed in a keystore file *and* the private key and signing certificate for the certificate issuer (CA) installed in a truststore file. The following figure illustrates these requirements and the relationship between the two files.

Example Truststore File and Keystore File Showing Relationship



As shown in the above figure, the same truststore file can contain multiple trusted root certificates (public keys for the signing CAs). These trusted roots might be associated with numerous keystore files. A keystore file can contain the key pairs for multiple CentraSite services, and the entire certificate chain required for a service's authentication.

With a certificate chain, it is necessary to validate each subsequent signature in the list until a trusted CA certificate is reached. For CentraSite, you must include the entire chain of certificates in a keystore and truststore. Also, any root CA certificates in use by clients must be in a CentraSite truststore.

Protecting Keystore and Truststore Files

Keystore and truststore files exist within the file system of your operating system, and since these are critically important files, you want to maintain them in a secure directory path. If either of these files cannot be located, CentraSite authentication will be disabled and no connection with the CentraSite can be made. It is recommended that only users serving as CentraSite administrators have access to these certificate files.

Configuring CentraSite to Use SSL

The configuration settings covered in this section deal with the CentraSite client side.

You can configure CentraSite client to use SSL in any of the following ways.

Configure CentraSite Client to Use One-way SSL

You perform the following procedure to configure CentraSite for one-way SSL authentication:

To configure one-way SSL

1. Create at least one truststore `centrasitetruststore.jks`, in JKS format, in a desired location on the machine where CentraSite is running.
2. Import the Mediator's self-signed certificate `mediator.cer` into the above created truststore or JAVA cacerts.

When prompted for password, the default for truststores is `password`.

```
C:\deploykeystores\new>keytool -export -alias mediator
-keystore mediatorkeystore.jks -rfc -file mediator.cer
Enter keystore password:
Certificate stored in file <mediator.cer>

C:\deploykeystores\new>keytool -import -alias mediator
-keystore centrasitetruststore.jks -file mediator.cer
Enter keystore password:
Re-enter new password:
Owner:
Issuer:
Serial number:
Valid from:
Certificate fingerprints:
    Trust this certificate? [no]: yes
Certificate was added to keystore

C:\deploykeystores\new>
```

If opting to import certificate in to Java cacerts, the Java runtime needs to trust the certificates of the Mediator in order to establish the SSL connections. To do that, add the certificate to the trusted certificates of Java via the `keytool` utility that comes with Java. The following command will add the certificate located at a location (for example, `c:\temp\server.crt`) to the trusted certificates in the Java used by CentraSite:

```
keytool.exe -import -v -trustcacerts -alias test -file "C:\temp\server.crt"
-keystore "<JDKInstallDir>\jre\lib\security\cacerts"
```

When prompted for password, the default for Java is `changeit`.

3. Add the following Java system properties to the `custom_wrapper.conf` file in `<SuiteInstallDir>/profiles/CTP/configuration` folder. For information about setting Java system properties, see the webMethods cross-product document, *Software AG Infrastructure Administrator's Guide*.

```
wrapper.java.additional.<n>=-Djavax.net.ssl.trustStore=
<location_of_truststore>
wrapper.java.additional.<n>=-Djavax.net.ssl.trustStorePassword=
<password_for_truststore>
```

In the settings above:

- `<n>` is a unique sequence number that you assign to each `wrapper.java.additional` property. For more information about assigning this sequence number, see the `wrapper.java.additional` property description in the cross-product document, *Working with the webMethods Product Suite and the Java Service Wrapper*.
 - `<location_of_truststore>` is the location to the trust store file (for example, `C:/deploykeystores/new/centrasitetruststore.jks`).
 - `<password_for_truststore>` is the password for the trust store.
4. Go to the section **#Java Additional Parameters**. Add the following property lines:


```
wrapper.java.additional.7=-Djavax.net.ssl.  
  
trustStore="C:/deploykeystores/new/centrasitetruststore.jks"  
wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=password
```
 5. Set the values as needed:


```
wrapper.java.additional.7=-Djavax.net.ssl.trustStore= represents the  
location of a truststore file (for example, centrasitetruststore.jks).  
  
wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=  
represents the password for a truststore.
```
 6. Save and close the file.
 7. Now restart the CentraSite Tomcat. All communication via the Mediator to the database should now be using SSL.

Configure CentraSite Client to Use Two-way SSL

You perform the following procedure to configure CentraSite for two-way SSL authentication:

To configure two-way SSL

1. Using OpenSSL, create a self-signed certificate (`centrasite.cer`) with the following command:

```
openssl req -new -x509 -days 2000 -sha1 -newkey rsa:1024 -nodes  
-keyout server.key -out server.crt -subj "/O=Company/OU=Unit/CN=localhost"
```

Whatever is specified in the CN section of the subject must match the hostname of the machine running the Mediator and is used to send requests to the Mediator.

2. Create at least one keystore `centrasitekeystore.jks`, in PKCS#12 or JKS format, containing a CentraSite key pair to use for SSL.

```
C:\deploykeystores\new>keytool -v -genkeypair -alias centrasite  
-keyalg RSA -validity 1000 -keystore centrasitekeystore.jks  
Enter keystore password:  
Re-enter new password:  
What is your first and last name?  
What is the name of your organizational unit?  
What is the name of your organization?  
What is the name of your City or Locality?  
What is the name of your State or Province?
```

```

What is the two-letter country code for this unit?

Enter key password for <centrasite>
    <RETURN if same as keystore password>:
[Storing centrasitekeystore.jks]

C:\deploykeystores\new>

```

3. Create at least one truststore `centrasitetruststore.jks`, in JKS format, in a desired location on the machine where CentraSite is running.
4. Import the Mediator's self-signed certificate `mediator.cer` into the above created truststore or Java cacerts.

When prompted for password, the default for truststores is `password`.

```

C:\deploykeystores\new>keytool -export -alias mediator
    -keystore mediatorkeystore.jks -rfc -file mediator.cer
Enter keystore password:
Certificate stored in file <mediator.cer>

C:\deploykeystores\new>keytool -import -alias mediator
    -keystore centrasitetruststore.jks -file mediator.cer
Enter keystore password:
Re-enter new password:
Owner:
Issuer:
Serial number:
Valid from:
Certificate fingerprints:
    Trust this certificate? [no]: yes
Certificate was added to keystore

C:\deploykeystores\new>

```

If opting to import certificate in to Java cacerts, the Java runtime needs to trust the certificates of the Mediator (regardless of whether this is a Tomcat application or a standalone application) in order to establish the SSL connections. To do that, add the certificate to the trusted certificates of Java via the `keytool` utility that comes with Java. The following command will add the certificate located at a location (for example, `c:\temp\server.crt`) to the trusted certificates in the Java used by CentraSite:

```

keytool.exe -import -v -trustcacerts -alias test
-file "C:\temp\server.crt"
-keystore "<JDKInstallDir>\jre\lib\security\cacerts"

```

When prompted for password, the default for Java is `changeit`.

5. Export the CentraSite's self-signed certificate `centrasite.cer` in to the Mediator's truststore.
6. Open the `wrapper.conf` file located in `<CentraSiteInstallDir>/profiles/CTP/configuration`
`<CentraSiteInstallDir>/profiles/CTP/configuration`
7. Go to the section `#Java Additional Parameters`. Add the following property lines:

```

wrapper.java.additional.5=-Djavax.net.ssl.keyStore="C:/deploykeystores/new/centrasitekeystore.jks"
wrapper.java.additional.6=-Djavax.net.ssl.keyStorePassword=password
wrapper.java.additional.7=-Djavax.net.ssl.trustStore="C:/deploykeystores/ne

```

```
w/centrasitetruststore.jks"
wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=password
```

8. Set the values as needed:

wrapper.java.additional.5=-Djavax.net.ssl.keyStore= represents the location of a keystore file (say, centrasitekeystore.jks).

wrapper.java.additional.6=-Djavax.net.ssl.keyStorePassword= represents the password for a keystore.

wrapper.java.additional.7=-Djavax.net.ssl.trustStore= represents the location of a truststore file (say, centrasitetruststore.jks).

wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword= represents the password for a truststore.

9. Save and close the file.

10. Now restart the CentraSite Tomcat. All communication via the Mediator to the database should now be using SSL.

Using the CTP Server.xml File for SSL

To configure SSL using CTP server.xml file

1. Open the server.xml file located in the following directory:

<CentraSiteInstallDir>/profiles/CTP/configuration/tomcat/conf

2. Enter the keystore information as specified below:

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Server>
3    <Service name="Catalina">
4      <Connector acceptCount="100" connectionTimeout="20000" description="CTP HTTP port" disableUploadTimeout="true" enableLook
5      <Connector SSLEnabled="true" acceptCount="100" algorithm="SunX509" clientAuth="false" description="CTP HTTPS port"
6      disableUploadTimeout="true" enableLookups="false"
7      keystoreFile="C:/deploykestores/new/centrasitekeystore.jks" keystorePass="password" keystoreType="JKS"
8      maxHttpHeaderSize="8192" maxSpareThreads="75" maxThreads="150" minSpareThreads="25" port="10011" scheme="https"
9      secure="true" sslProtocol="TLS" />
10     <Engine defaultHost="localhost" name="Catalina">
11       <Host appBase="C:\SoftwareAG\profiles\CTP\workspace\webapps" autoDeploy="true" name="localhost" unpackWARs="true" workD:
12     </Engine>
13     <Connector description="CentraSite 8.2" maxHttpHeaderSize="8192" maxThreads="150" minSpareThreads="25" maxSpareThreads="7
14     <!--for configuring SSL s. comment for Connector for port 49982 above-->
15     <Connector description="CentraSite 8.2 HTTPS" maxHttpHeaderSize="8192" maxThreads="150" minSpareThreads="25" maxSpareThre
16   </Service>
17 </Server>
18
19
```

3. Add the Mediator certificate (mediator.cer) into CentraSite JVM cacerts as below:

```

C:\WINDOWS\system32\cmd.exe
C:\deploykeystores\new>keytool -import -alias mediator -keystore C:\Software\G\jun\jun160_32\jre\lib\security\cacerts -file mediator.cer
Enter keystore password:
Owner: CN=pcinriya.eur.ad.sag, OU=Mediator, O=WebMethods, L=Reston, ST=VA, C=US
Issuer: CN=pcinriya.eur.ad.sag, OU=Mediator, O=WebMethods, L=Reston, ST=VA, C=US
Serial number: 45a57bd3
Valid from: Fri Oct 01 11:42:35 IST 2010 until: Thu Jun 27 11:42:35 IST 2013
Certificate fingerprints:
    MD5: E2:0D:90:2E:E4:5E:0E:00:3A:EC:72:53:47:33:98:E0
    SHA1: 4F:A1:CB:8A:50:E9:BA:B2:B0:61:CD:D3:43:3F:CB:9F:5A:07:F7:F2
Signature algorithm name: SHA1withRSA
Version: 3
Trust this certificate? [no]: yes
Certificate was added to keystore
C:\deploykeystores\new>

```

Configuring webMethods Integration Server to Use SSL

The configuration settings covered in this section deal with the webMethods Integration Server side.

You configure an Integration Server to use one- or two-way SSL.

Configure Integration Server to Use One-way SSL

You perform the following procedure to configure Integration Server for one-way SSL authentication:

To configure one-way SSL

1. Using OpenSSL, create a self-signed certificate (mediator.cer) with the following command:

```

openssl req -new -x509 -days 2000 -sha1 -newkey rsa:1024 -nodes
-keyout server.key -out server.crt -subj "/O=Company/OU=Unit/CN=localhost"

```

Whatever is specified in the CN section of the subject must match the hostname of the machine running the Mediator and is used to send requests to Mediator.

2. Create at least one keystore mediatorkeystore.jks, in PKCS#12 or JKS format, containing an Integration Server key pair to use for SSL and its corresponding key alias.

```

C:\deploykeystores\new>keytool -v -genkeypair -alias mediator
-keyalg RSA -validity 1000 -keystore mediatorkeystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
What is the name of your organizational unit?
What is the name of your organization?
What is the name of your City or Locality?
What is the name of your State or Province?
What is the two-letter country code for this unit?

Enter key password for <mediator>
    <RETURN if same as keystore password>:
[Storing mediatorkeystore.jks]

C:\deploykeystores\new>

```

3. Export the Mediator's self-signed certificate mediator.cer into the CentraSite's truststore.

4. Configure an HTTPS port and specify the client authentication to **Username/Password**. The server prompts the client for a user ID and password.
5. On the **Ports** screen, click **Edit** to change the **Access Mode**. You may Set Access Mode to **Allow by Default** or **Reset to default access settings**.

For more information on configuring ports and client authentication, see *webMethods Integration Server Administrator's Guide*.

6. Restart the Integration Server.

Configure Integration Server to Use Two-way SSL

You perform the following procedure to configure Integration Server for one-way SSL authentication:

To configure two-way SSL

1. Using OpenSSL, create a self-signed certificate (mediator.cer) with the following command:

```
openssl req -new -x509 -days 2000 -sha1 -newkey rsa:1024 -nodes
-keyout server.key -out server.crt -subj "/O=Company/OU=Unit/CN=localhost"
```

Whatever is specified in the CN section of the subject must match the hostname of the machine running the Mediator and is used to send requests to the Mediator.

2. Create at least one keystore mediatorkeystore.jks, in PKCS#12 or JKS format, containing an Integration Server key pair to use for SSL.

```
C:\deploykeystores\new>keytool -v -genkeypair -alias mediator
-keyalg RSA -validity 1000 -keystore mediatorkeystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
What is the name of your organizational unit?
What is the name of your organization?
What is the name of your City or Locality?
What is the name of your State or Province?
What is the two-letter country code for this unit?

Enter key password for <mediator>
    <RETURN if same as keystore password>:
[Storing mediatorkeystore.jks]

C:\deploykeystores\new>
```

3. Create at least one truststore mediatortruststore.jks, in JKS format, in a desired location on the machine where CentraSite is running.
4. Export the Mediator's self-signed certificate mediator.cer into the CentraSite's truststore.
5. Import the CentraSite's self-signed certificate centrasite.cer in to the Mediator's truststore mediatortruststore.jks.

```
C:\deploykeystores\new>keytool -export -alias
centrasite -keystore centrasitekeystore.jks -rfc -file
```

```

centrasite.cer
Enter keystore password:
Certificate stored in file <centrasite.cer>

C:\deploykeystores\new>keytool -import -alias
mediator -keystore mediatortruststore.jks -file
centrasite.cer
Enter keystore password:
Re-enter new password:
Owner:
Issuer:
Serial number:
Valid from:
Certificate fingerprints:
                Trust this certificate? [no]: yes
Certificate was added to keystore

C:\deploykeystores\new>

```

6. Create a keystore and truststore alias using the above created keystore (mediatorkeystore.jks) and truststore (mediatortruststore.jks) respectively. For more information on creating keystore and truststore aliases, see *webMethods Integration Server Administrator's Guide* in the documentation set for webMethods Integration Server.
7. Configure an HTTPS port and specify the client authentication to any of the following:
 - **Username/Password.** The server prompts the client for a user ID and password.
 - **Request Client Certificates.** The server requests client certificates for all requests. If the client does not provide a certificate, the server prompts the client for a userid and password. If the client provides a certificate:
 - The server checks whether the certificate exactly matches a client certificate on file and is signed by a trusted authority. If so, the client is logged in as the user to which the certificate is mapped in Integration Server. If not, the client request fails, unless central user management is configured.
 - If central user management is configured, the server checks whether the certificate is mapped to a user in the central user database. If so, the server logs the client on as that user. If not, the client request fails.
 - **Require Client Certificates.** The server requires client certificates for all requests. The server behaves as described for Request Client Certificates, except that the client must always provide a certificate.
8. On the **Ports** screen, click **Edit** to change the **Access Mode**. You may Set Access Mode to Allow by Default or Reset to default access settings.
9. If chosen client authentication as **Require Client Certificates** above, map the client certificate to any valid user in the Integration Server.

For more information on configuring ports and client authentication, see *webMethods Integration Server Administrator's Guide*.

10. Restart the Integration Server.

Configuring webMethods Mediator to Use SSL

Configure your instance of webMethods Mediator as described in *Administering webMethods Mediator*.

6 Virtualized APIs in CentraSite Business UI

■ Introduction to Virtualized APIs	188
■ Roles and Permissions Needed to Create and Manage Virtualized APIs	188
■ Preparing to Virtualize and Publish the Virtualized APIs	189
■ Run-Time Components of Virtualized API	190
■ Creating Virtualized APIs	203
■ Reconfiguring a Virtualized API	211
■ Resource Synchronization in Virtual REST APIs	214
■ Versioning Virtual API or Service	219
■ Viewing or Changing a Virtualized API	223
■ Assigning Run-Time Actions to a Virtualized API	228
■ Publishing and Unpublishing APIs to and from Gateways	231
■ Exposing a Virtual SOAP API as Virtual REST API	244
■ Displaying Runtime Information for a Virtualized API	245
■ Obtaining Your API Keys and Access Tokens for Consumption	250
■ Managing Your API Keys	254
■ Privileged User of a Virtualized API	260

Introduction to Virtualized APIs

API Providers (Asset Providers) and API Consumers (Asset Consumers) use the CentraSite Business UI to browse and search for virtualized APIs in the asset catalog by name, description, attribute values, asset types and/or taxonomy groups. Additionally, the API Providers can virtualize and publish APIs to the run-time layer; while the API consumers can invoke (call) the virtualized APIs that are exposed to them.

This topic describes how to:

- Create and configure virtualized APIs in the CentraSite catalog.
- Publish virtualized APIs to the webMethods Mediator and webMethods API-Portal gateways.
- Consume APIs using the API key or OAuth2 client access token.

Instructions throughout the remainder of this document use the term "API" when referring to the instances of the three base types (Service, XML Service, and REST Service), and "Virtualized API" when referring to the instances of the three virtual types (Virtual Service, Virtual XML Service, and Virtual REST Service) in general.

Roles and Permissions Needed to Create and Manage Virtualized APIs

To create and manage virtualized APIs, you must have the following roles or permissions:

- CentraSite Administrator
- Organization Administrator
- Asset Provider
- API Runtime Provider (required to configure run-time actions for the virtualized APIs)
- Mediator Publisher (required to publish virtualized APIs to Mediator gateways)
- API-Portal Publisher (required to publish virtualized APIs to API-Portal gateways)
- Instance-level Modify permission for a gateway (required to publish virtualized APIs to that particular gateway)

If you have the CentraSite Administrator role, you can create and manage virtualized APIs within any organization.

If you have the Organization Administrator role or API-Portal Administrator role for a specific organization, you have the ability to create and manage virtualized APIs within that specific organization.

For more information about roles and permissions, see *Getting Started with CentraSite*.

Preparing to Virtualize and Publish the Virtualized APIs

Before API providers can publish the virtualized APIs to Mediator and API-Portal gateways, some additional steps are required. The following table lists these high-level steps and where to go for more information:

Step	Where to Go for Procedures
<p>(Specific for API-Portal) Configure design-time and change-time policies using the predefined policies Publish to API-Portal and Unpublish from API-Portal.</p> <p>This step is needed only if your organization requires design-time governance.</p>	<p>Section on the design/change-time policies for API-Portal in <i>Working with the CentraSite Business UI</i></p>
<p>Set up approval and onboarding workflows in CentraSite.</p>	<p>Section on the predefined policies for API management in <i>Working with the CentraSite Business UI</i></p>
<p>Create a native API in CentraSite.</p>	<p>Sections on creating a Native SOAP-based API in <i>Working with the CentraSite Business UI</i> and Native REST API in <i>Working with REST-based APIs in CentraSite</i></p>
<p>Create a virtual proxy alias for the native API in CentraSite.</p>	<p>Section on creating a virtualized API in "Creating Virtualized APIs" on page 203</p>
<p>Configure run-time enforcement for the virtualized API you want to expose.</p>	<p>Section on assigning the policy actions to the virtualized API in "Assigning Run-Time Actions to a Virtual API" on page 228</p>
<p>Ensure that the run-time policies for the virtualized API are active.</p>	<p>Section on activating a run-time policy in <i>Working with the CentraSite Business UI</i></p>
<p>Configure consumption settings for the native API.</p>	<p>Section on configuring the native API's consumption settings in API key management policies in</p>

Step	Where to Go for Procedures
	<i>Working with the CentraSite Business UI</i>
Create an instance of Mediator gateway, for example, webMethods Mediator in CentraSite.	Section on registering a Mediator instance with CentraSite in "Creating and Managing Gateways" on page 61
Create an instance of API-Portal gateway, for example, webMethods API-Portal in CentraSite.	Section on registering an API-Portal instance with CentraSite in <i>Working with the CentraSite Business UI</i>
Ensure that the Mediator gateway's specified deployment URL is active and the user credentials of Integration Server are valid.	Section on creating and managing Mediator gateways in "Creating and Managing Gateways" on page 61
Publish the virtualized API to Mediator and webMethods API-Portal gateways.	Section on publishing the native API and/or virtualized APIs to the Mediator and webMethods API-Portal gateways in "Publishing and Unpublishing APIs to and from Gateways" on page 231

If these conditions are not satisfied, all or part of the publish user interface controls will be disabled when you view a particular API in CentraSite Business UI.

Run-Time Components of Virtualized API

The run-time behavior of a virtualized API is defined by the following components:

- Policy Accordions
- Run-Time Actions
- Message Flow Stages

Summary of the Policy Accordions

The policy actions are classified into one of the following accordions:

Accordion	Use to...
Request Handling	Request handler allows receiving and transforming the incoming message from a client into a custom format as

Accordion	Use to... expected by the native API. For example, a Require HTTP / HTTPS action mandates that the incoming requests for an API are received over the HTTP and/or HTTPS protocol.
Policy Enforcement	Enforces the adherence to real-time policy compliance identifying/authenticating, monitoring, auditing, and measuring and collecting result statistics for the virtualized API.
Response Processing	Response handler allows processing of the response message coming from the native API into a custom format as expected by the client.
Error Handling	Error handlers are used to format and return error messages. Errors can occur at run-time for various reasons. For example, security errors occur if a username is not correctly validated or authorized; transformation errors occur if transformation action is unable to successfully transform a message; a routing error is raised if a routing endpoint is unavailable, and so on.

Summary of the Run-Time Actions

Actions are the core elements of stages in a message flow that define the handling of messages as they flow through a virtualized API. The following table lists the actions you can configure for the virtualized API's message flow, and links you to topics that describe the actions, including how to configure them.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
Request Handling > Protocol			
Require JMS	Specify the JMS protocol for the API to accept and process the requests.	Receive	<ul style="list-style-type: none"> ■ SOAP APIs.
Require HTTP / HTTPS	Specify the HTTP and/or HTTPS protocol and the SOAP format (for a SOAP-based API) for the API to	Receive	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	accept and process the requests.		
Request Transformation	Invoke an XSL transformation in the incoming request before it is submitted to the an API.	Receive	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Invoke webMethods Integration Server	Invoke a webMethods Integration Server service to pre-process the request before it is submitted to the an API.	Receive	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Enable REST Support	Enables REST support for an existing SOAP based API by exposing the API both as a REST based API as well as a SOAP API.	Receive	<ul style="list-style-type: none"> ■ SOAP APIs.
Set Media Type	Specifies the content type for a REST request received from a client if the content type header is not specified.	Receive	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Policy Enforcement > Authentication			
HTTP Basic Authentication	Identify and validate the consumer's authentication credentials	Routing	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	contained in the request's Authorization header using HTTP basic authentication mechanism.		
NTLM Authentication	Identify and validate the consumer's authentication credentials contained in the request's Authorization header using NTLM authentication mechanism.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
OAuth2 Authentication	Identify and validate the consumer's authentication credentials contained in the request's Authorization header using OAuth 2.0 authentication mechanism.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Policy Enforcement > JMS Routing			
JMS Routing Rule	Specify a JMS queue to which the Mediator is to submit the request, and the destination to which the an API	Routing	<ul style="list-style-type: none"> ■ SOAP APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	is to return the response.		
Set Message Properties	Specify JMS message properties to authenticate client requests before submitting to the APIs.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs.
Set JMS Headers	Specify JMS headers to authenticate client requests before submitting to the APIs.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs.
Policy Enforcement > Logging and Monitoring			
Log Invocation	Log request/response payloads to a destination you specify.	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Monitor Service Performance	Monitor the run-time performance for a specific consumer, and defines the level of performance that the specified consumer should expect from the API.	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Monitor Service Level Agreement	Monitor a user-specified set of run-time performance conditions for an API, and sends alerts to a specified	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	destination when these performance conditions are violated.		
Policy Enforcement > Routing			
Straight Through Routing	Route requests directly to a native endpoint that you specify.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Content Based Routing	Route requests to different endpoints based on specific values that appear in the request message.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Load Balancing and Failover Routing	Routes the requests across multiple endpoints.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Context Based Routing	Route requests to different endpoints based on specific values that appear in the request message.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Set Custom Headers	Specify the HTTP headers to process the requests.	Routing	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Policy Enforcement > Security			
Require SSL	Mandate that requests be sent via SSL client certificates, and can be used by	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	both SOAP and REST APIs.		
Require Signing	Mandate that a request's XML element (which is represented by an XPath expression) be signed.	Enforce	■ SOAP APIs.
Require Encryption	Mandate that a request's XML element (which is represented by an XPath expression) be encrypted.	Enforce	■ SOAP APIs.
Require Timestamps	Mandate that timestamps be included in the request header.	Enforce	■ SOAP APIs.
Require WSS SAML Token	Uses a WSS Security Assertion Markup Language (SAML) assertion token to validate API consumers.	Enforce	■ SOAP APIs.
Evaluate HTTP Basic Authentication	■ Identify the consumer against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available users in Mediator).	Enforce	■ SOAP APIs. ■ REST APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	<ul style="list-style-type: none"> ■ Validate the client's authentication credentials contained in the request's Authorization header against the list of users in the Integration Server on which Mediator is running. 		
Evaluate Hostname	<ul style="list-style-type: none"> ■ Identify the consumer against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available users in Mediator). ■ Validate the client's IP address against the list of users in the Integration Server on which Mediator is running. 	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Evaluate IP Address	<ul style="list-style-type: none"> ■ Identify the consumer against either the Registered Consumers list (the list of registered 	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	<p>consumers in Mediator) or the Global Consumers list (the list of available users in Mediator).</p> <ul style="list-style-type: none"> ■ Validate the client's IP address against the list of users in the Integration Server on which Mediator is running. 		
Evaluate WSS Username Token	<ul style="list-style-type: none"> ■ Identify the consumer against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available users in Mediator). ■ Validate the client's WSS username token against the list of users in the Integration Server on which Mediator is running. 	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs.
Evaluate WSS X.509 Certificate	<ul style="list-style-type: none"> ■ Identify the consumer against either the Registered 	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	<p>Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available users in Mediator).</p> <ul style="list-style-type: none"> ■ Validate the client's WSS X.509 token against the list of users in the Integration Server on which Mediator is running. 		
Evaluate XPath Expression	<ul style="list-style-type: none"> ■ Identify the consumer against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available users in Mediator). ■ Validate the client's XPath expression against the list of users in the Integration Server on which Mediator is running. 	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
Evaluate OAuth2 Token	<ul style="list-style-type: none"> ■ Identify the consumer against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available users in Mediator). ■ Validate the client's IP address against the list of users in the Integration Server on which Mediator is running. 	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Evaluate Client Certificate for SSL Connectivity	<ul style="list-style-type: none"> ■ Identify the consumer against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available users in Mediator). ■ Validate the client's certificate against the list of users in the Integration Server on which 	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
	Mediator is running.		

Policy Enforcement > Traffic Management

Throttling Traffic Optimization	<ul style="list-style-type: none"> ■ Limit the number of API invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated. ■ Avoid overloading the back-end services and their infrastructure, to limit specific consumers in terms of resource usage, and so on. 	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
---	---	---------	--

Service Result Cache	Enables caching of the results of SOAP and REST API invocations.	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
--------------------------------------	--	---------	--

Policy Enforcement > Validation

Service Result Cache	Validate all XML request and/or response messages against an XML schema referenced in the WSDL.	Enforce	<ul style="list-style-type: none"> ■ SOAP APIs.
--------------------------------------	---	---------	--

Response Processing

This action...	Use to...	Available in Message Flow Stage...	Applicable for...
Response Transformation	Invoke an XSL transformation in the SOAP response payloads from XML format to the format required by the consumer.	Response	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Invoke webMethods Integration Server	Invoke a webMethods Integration Server service to process the response from the an API before it is returned to the consumer.	Response	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Set Media Type	Specifies the content type for a REST response.	Response	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.
Error Handling			
Conditional Error Processing	Return a custom error message (and/or the native provider's service fault content) to the consumer when the native provider returns a service fault.	Response	<ul style="list-style-type: none"> ■ SOAP APIs. ■ REST APIs.

For detailed descriptions about the individual run-time actions that CentraSite out-of-the-box, see "[Built-In Run-Time Actions Reference for Virtual Services](#)" on page 347.

Summary of the Message Flow Stages

Message Flow defines the implementation of a virtualized API.

A message flow is a sequence of stages representing a non-branching one-way processing path. Message flow is used for request, enforce, routing, and response paths as well as for error handlers.

A message flow tree is constructed by chaining together actions of these top-level stages:

Stage	Description
Receive	Receive stage is used for processing the request path of the Message Flow.
Enforce	Enforce stage is used for processing the enforcement path of the Message Flow. The enforce stage is used to identify and validate specific consumers invoking APIs, throttle traffic, log request/response payloads, and monitor run-time performance conditions.
Routing	Routing stage is used for processing the routing path of the Message Flow. The routing stage is used to perform request-response communication. It represents the boundary between request and response processing for the API. When the routing stage dispatches a request message, request processing is considered finished. When the routing stage receives a response message, response processing begins.
Response	Response stage is used for processing the response path of the Message Flow. In addition, response stage is used as error handler.

Creating Virtualized APIs

Before You Begin

To virtualize and publish a Virtual API, the following prerequisites must be met:

- Make sure that you have the roles and permissions required for creating virtualized APIs as specified in ["Roles and Permissions Needed to Create and Manage Virtualized APIs" on page 188](#).
- Ensure that the webMethods Mediator is configured and running on the same machine webMethods Integration Server.
- *Applicable for OAuth2 scenarios.* Ensure that the webMethods Mediator property `watt.server.auth.skipForMediator` is set to `true`. For details, see ["The watt.server.auth.skipForMediator Property" on page 350](#).

Ways in Which You Can Create Virtualized APIs

You can create a virtualized API (that is, add a virtualized API) to the CentraSite registry in the following three ways:

- You can create a virtualized API from an existing API, also called as a Native Service in CentraSite, meaning that you create the virtual copy (proxy) of the existing API using an already imported input file. For procedures, see ["Creating a Virtualized API from an Existing Native API" on page 204](#).
- You can create a virtualized API from scratch, meaning that you create the virtualized API (and set its attributes) manually. For procedures, see ["Creating a Virtualized API from Scratch" on page 207](#).
- You can create a virtualized API using an importer, which is a utility that generates an appropriate virtualized API from an imported archive file. For more information about using an importer, refer to the section "Importing Assets" in *Working with the CentraSite Business UI*.

Creating a Virtualized API from an Existing Native API

To create a new virtualized API from an existing API, perform the following high-level steps:

Creating a Virtual SOAP API

To create a Virtual (SOAP-based) API, also called as a Virtual Service in CentraSite, from an existing Native (Web) Service, proceed as follows:

Add a Virtual SOAP API

You use the panel 1 of the Virtualize your API page to specify the proxy and invocation aliases, and endpoints for the new Virtual API.

To add a Virtual SOAP API

1. Display the list of Web services, by executing an advanced search for **Service** asset type (see *Working with the CentraSite Business UI* for details).
2. In the displayed list, click the link of the Web service for which you want to create a virtual copy. This shows the profile details of the Web service.
3. In the details page of the Web service, click the **Virtualize** icon.
4. In the **Virtualize <API_Name> (Step 1 of 3)** wizard, complete the fields.

In this field...

Create a New Virtual Alias

Do the following...

Enter a name for the new virtual alias.

This name must be NCName-conformant, meaning that:

- The name must begin with a letter or the underscore character (`_`).
- The remainder of the name can contain any combination of letters, digits, or the following

In this field...	Do the following...
	<p>characters: . - _ (that is, period, dash, or underscore). It can also contain combining characters and extender characters (for example, diacriticals).</p> <ul style="list-style-type: none"> ■ The name cannot contain any spaces. ■ Furthermore, if the Virtual API name contains any non-conformant character, upon publishing the Virtual API to any gateway, the non-conformant character is simply replaced with the underscore character (_) in Mediator. However, in CentraSite the Virtual API name defined by you is displayed. <p>For more information about the NCName type, see http://www.w3.org/TR/xmlschema-2/#NCName</p> <p>An API name does not need to be unique within the registry. However, if for example, a Virtual API with the same name already exists within the native API, a warning message will be issued.</p> <p>By default, CentraSite populates the Create a New Virtual Alias field with the display name that was specified for the native API.</p>
Endpoint prefix for invocation alias	<p><i>Optional.</i> Enter a prefix for the alias. It accepts URL paths. For example, /testmethod/myprefix/, test@1234, and so on.</p>
Endpoints of <API_name> to be virtualized	<p>Select the endpoint that you want to use for invoking the Virtual API.</p> <p>The Endpoints list displays the URLs of all endpoints that are available for the native API.</p>

5. Click **Next** to complete the settings.

Assign Run-Time Actions for the Virtual SOAP API

You use the panel 2 of the Virtualize your API page to configure the policy actions for the Virtual API.

1. In the **Virtualize <API_Name> (Step 2 of 3)** wizard, drag and drop the policy actions that you want CentraSite to execute for the Virtual API at run-time.
2. Configure the parameters for each action as described in "[Assigning Run-Time Actions to a Virtual API](#)" on page 228, and then click **Next**.

Implement Virtualization and Publish the Virtual SOAP API to Gateways

You use the panel 3 of the Virtualize your API page to configure the gateways for publishing the Virtual API. The publishing operation allows the API Providers to expose the Virtual API in the selected gateways. Clients can then access and examine the usage of the exposed Virtual API.

1. In the **Virtualize <API_Name> (Step 3 of 3)** wizard, display the list of available gateways that are available to you.
2. In the **Gateway** drop-down box, choose the gateways (for example, API-Portal, Mediator, or Insight) to publish the Virtual API.
3. In the **Sandbox** drop-down box, choose the category by which the gateways were classified.
4. Mark the checkbox of each gateway to which you want to publish the Virtual API.
5. Applicable for an API-Portal gateway only. Configure a set of sandbox categories for publishing the Virtual API to the specified sandbox categories of the API-Portal gateways. Do the following:
 - a. Click the **Configure** icon next to **Sandbox** field.
 - b. In the **API-Portal Settings** dialog box, select the sandbox category to which you want to publish the Virtual API.
6. Applicable for a Mediator/Insight gateway only. Select the **Expose to Consumers** option to allow clients to simply access and examine the usage of the exposed Virtual API.
7. Click **Virtualize** to create the virtual copy (proxy) of the Web Service (without publishing the newly created Virtual API to the selected gateways). This opens the details page of the newly created Virtual API.

The details page for the Virtual API that you just created is displayed.

- Else, click **Publish** to create the virtual copy of the Web Service and simultaneously publish the newly created Virtual API to the selected gateways.

Creating a Virtual REST API

To create a Virtual REST API, also called as a Virtual REST Service in CentraSite, from an existing Native REST Service, proceed as follows:

Add a Virtual REST API

You use the panel 1 of the Virtualize your API page to specify the proxy and invocation aliases, and endpoints for the new Virtual REST API.

1. Display the list of REST APIs, by executing an advanced search for **REST Service** asset type (see *Working with the CentraSite Business UI* for details).

2. In the displayed list, click the link of the REST API for which you want to create a virtual copy. This shows the profile details of the REST API.
3. In the actions bar of the REST API, click the **Virtualize** icon.
4. In the **Virtualize <API_Name> (Step 1 of 3)** wizard, complete the fields. The fields displayed for a Virtual REST API are the same as for the Virtual SOAP API, so for a description of the fields, follow the instructions provided above for "[Creating a Virtual SOAP API](#)" on page 204.
5. In the **Resources of <API_name> to Virtualize** field, select the resources you want to use for the Virtual REST API or select "**All**" if you want to use all of the available resources.
6. Click **Next** to complete the settings.

Assign Run-Time Actions for the Virtual REST API

You use the panel 2 of the Virtualize your API page to configure the policy actions for the Virtual REST API.

The fields displayed for a Virtual REST API are the same as for the Virtual API, so for a description of the fields, follow the instructions provided above for "[Creating a Virtual SOAP API](#)" on page 205.

Implement Virtualization and Publish the Virtual REST API to Gateways

You use the panel 3 of the Virtualize your API page to configure the gateways for publishing the Virtual REST API. The publishing operation allows the API Providers to expose the Virtual REST API in the selected gateways. Clients can then access and examine the usage of the exposed Virtual REST API.

The fields displayed for a Virtual REST API are the same as for the Virtual API, so for a description of the fields, follow the instructions provided above for "[Creating a Virtual SOAP API](#)" on page 206.

Creating a Virtualized API from Scratch

CentraSite allows you to add a Virtual SOAP API or Virtual REST API to the registry manually, even if you can create them using imported files. Be aware, however, that you might not be able to manually set all of the attributes for the Virtual Service or Virtual REST Service type.

To create a new virtualized API from scratch, perform the following high-level steps:

Creating a Virtual SOAP API

To create a Virtual (SOAP-based) API from scratch, proceed as follows:

Add a Virtual SOAP API

You use the Create New Asset page to add a new Virtual SOAP API to the registry.

1. In the activity bar, click **Create Asset**. This opens the Create New Asset page.
2. In the **Basic Information** section, complete the fields as necessary.

<u>In this field...</u>	<u>Do the following...</u>
Name	<p>(Mandatory) Enter a name for the Virtual SOAP API. A Virtual API name can contain any character (including spaces).</p> <p>The Virtual SOAP API name does not need to be unique within the catalog. However, to reduce ambiguity, you should avoid giving multiple Virtual SOAP APIs of the same type "Service" the same name. As a best practice, we recommend that you adopt appropriate naming conventions to ensure that Virtual SOAP APIs are distinctly named.</p>
Type	Choose Virtual Service from the drop-down list in the dialog box.
Organization	<p>Choose an organization from the drop-down list in which you want to create Virtual SOAP APIs.</p> <p>Note: The drop-down list shows only the organizations to which you are permitted to create Virtual SOAP APIs.</p> <p>If you specify an organization, the newly created Virtual SOAP API will be created in this organization. If you do not specify an organization, the Virtual API is created in the Default Organization.</p>
Version	(Optional) Enter an identifier for the initial version of the Virtual SOAP API.
Description	(Optional) Enter a comment or descriptive information about the Virtual SOAP API.
Import a File	<p>Specify whether the input file will be read from your local file system (the File option) or from a URL-addressable location on the network (the URL option).</p> <p>If the file you are importing resides on the network, specify its URL.</p> <p>If the file resides in your local file system, specify the file name. You can use the Browse button to navigate to the required folder.</p>
Advanced Settings	In the Advanced Settings node, do the following:

In this field...	Do the following...
	<ul style="list-style-type: none"> ■ Credentials - If you have specified a URL, and the site you want to access via the URL requires user authentication, enter a username and password for authentication at the URL site. ■ Resolution - Choose a resolution strategy, which will allow you to specify how an already existing imported/included file is handled. For each of the imported/included files you have one of these options: <ul style="list-style-type: none"> ■ Overwrite the importing file with new content. ■ Create a new version of the file with the new content (if, for example, you want to modify a WSDL file but want to retain its previous version).

3. Click **Next** to leave the Create New Asset page.
4. In the **Preview** panel, review the basic information for the Virtual SOAP API before you actually add to the CentraSite registry.
5. Click **Save** to add the new Virtual SOAP API to CentraSite.

The details page for the Virtual API that you just created is displayed.

Upload an Input File

Certain attributes such as the list of operations and endpoints cannot be specified manually. To set these attributes, you must attach the WSDL file to the Virtual SOAP API. If you have not specified the WSDL file using the **Import a File** field in the Create New Asset page, you can use the **Attach** action in the API's actions bar to upload an input WSDL file. The Attach Document operation allows CentraSite to automatically populate the applicable SOAP operations and endpoints for the Virtual SOAP API.

1. In the actions bar of the Virtual SOAP API, click **Attach**.
2. In the **Attach Document** dialog, complete the fields.
3. Click **Attach**.

Assign Run-Time Actions to the Virtual SOAP API

You use the **Virtualize** action to configure the run-time policy actions for the Virtual SOAP API.

1. In the actions bar of the Virtual SOAP API, click **Virtualize**.
2. In the **Virtualize <API_Name> (Step 1 of 2)** wizard, drag and drop the policy actions that you want CentraSite to execute for the Virtual SOAP API at run-time.
3. Configure the parameters for each action as described in "[Assigning Run-Time Actions to a Virtual API](#)" on page 205, and then click **Virtualize**.

Publish the Virtual SOAP API to Gateways

You use the **Publish** action to configure the gateways for publishing the Virtual SOAP API. The publishing operation allows the API Providers to expose the Virtual SOAP API in the selected gateways. Clients can then access and examine the usage of the exposed Virtual SOAP API.

1. In the actions bar of the Virtual SOAP API, click **Publish**.
2. In the **Publish** dialog, display the list of available gateways that are available to you.
3. In the **Gateway** drop-down box, choose the gateways (for example, API-Portal, Mediator, or Insight) to publish the Virtual API.
4. In the **Sandbox** drop-down box, choose the category by which the gateways were classified.
5. Mark the checkbox of each gateway to which you want to publish the Virtual SOAP API.
6. Applicable for an API-Portal gateway only. Configure a set of sandbox categories for publishing the Virtual SOAP API to the specified sandbox categories of the API-Portal gateways. Do the following:
 - a. Click the **Configure** icon next to **Sandbox** field.
 - b. In the **API-Portal Settings** dialog box, select the sandbox category to which you want to publish the Virtual SOAP API.
7. Applicable for a Mediator/Insight gateway only. Select the **Expose to Consumers** option to allow clients to simply access and examine the usage of the exposed Virtual SOAP API.
8. Click **Publish** to publish the newly created Virtual API to the selected gateways.

Creating a Virtual REST API

To create a Virtual (REST) API from scratch, proceed as follows:

Add a Virtual REST API

You use the Create New Asset page to add a new Virtual REST API to the registry.

1. In the activity bar, click **Create Asset**. This opens the Create New Asset page.
2. In the **Basic Information** section, complete the fields. The fields displayed for a Virtual REST API are the same as for the Virtual API, so for a description of the fields, follow the instructions provided above for ["Creating a Virtual SOAP API" on page 207](#).
3. Click **Next** to leave the Create New Asset page.
4. In the **Preview** panel, review the basic information for the Virtual REST API before you actually add to the CentraSite registry.
5. Click **Save** to add the new Virtual REST API to CentraSite.

The details page for the Virtual REST API that you just created is displayed.

Add Resources and Methods

You use the **Resources and Methods** profile to add the REST resources and methods for the Virtual REST API.

If you do not want to add the REST resources and methods manually, CentraSite provides the REST importer that reads a RAML or Swagger input file and automatically adds the resources and methods to the Virtual REST API. For information about using REST importers, see *Working with REST-based APIs in CentraSite*.

1. In the actions bar of the Virtual REST API, click the **Edit** icon.
2. Click the **Resource and Methods** profile.
3. Click on the **Add Resource** link.
4. In the **Add Resource** dialog box, complete the fields as necessary. For information about adding a REST resource, see *Working with REST-based APIs in CentraSite*.
5. Expand the resource node to that you want to add a HTTP method.
6. Click on the **Add Method** link.
7. In the **Add Method** dialog box, complete the fields as necessary. For information about adding a REST method, see *Working with REST-based APIs in CentraSite*.
8. Click **Save** to save the details.

Assign Run-Time Actions to the Virtual REST API

You use the **Virtualize** action to configure the run-time policy actions for the Virtual REST API.

The fields displayed for a Virtual REST API are the same as for the Virtual API, so for a description of the fields, follow the instructions provided above for "[Creating a Virtual SOAP API](#)" on page 209.

Publish the Virtual REST API to Gateways

You use the **Publish** action to configure the gateways for publishing the Virtual REST API.

The fields displayed for a Virtual REST API are the same as for the Virtual API, so for a description of the fields, follow the instructions provided above for "[Creating a Virtual SOAP API](#)" on page 210.

Reconfiguring a Virtualized API

This section describes how to use the **Virtualize** action to reconfigure virtualized APIs that were created using an existing native API or from scratch in the CentraSite registry.

To access the **Virtualize** action in the details page of native API or virtualized API, you must have the following roles or permissions:

- CentraSite Administrator
- Organization Administrator
- Asset Provider
- Instance-level Modify permission on the native API or virtualized API

In addition to the above described roles and permissions for managing a particular virtualized API, make sure that you have the specific roles and permissions required for reconfiguring the run-time actions and republishing the virtualized APIs to Mediator and API-Portal gateways as specified in ["Roles and Permissions Needed to Create and Manage Virtualized APIs"](#) on page 188.

For details about roles and permissions, see *Getting Started with CentraSite*.

Ways in Which You Can Reconfigure Virtualized APIs

When you use the **Virtualize** action to reconfigure virtualized APIs, the **Virtualize <API_Name>** wizard is automatically populated depending on the way in which the **Virtualize** action is accessed. In CentraSite, you can reconfigure an existing virtualized API in the following two ways:

- Using the **Virtualize** action from the details page of the native API. This allows you to do the following:
 - Create a new virtualized API. For procedures, see ["Creating a Virtualized API from an Existing Native API"](#) on page 204.
 - Reconfigure an existing virtualized API. During this procedure, you reconfigure the existing virtualized API's endpoints, synchronize the REST resources, reconfigure the policy actions and publish to the gateways.
- Using the **Virtualize** action from the details page of the virtualized API. This allows you to reconfigure the virtualized API's policy actions and publish to the gateways.

Reconfiguring a Virtualized API from the Native API Details Page

To reconfigure a virtualized API from the details page of a particular native API, proceed as follows:

1. Display the list of native APIs (see *Working with the CentraSite Business UI* for details).
2. In the displayed list, click the link of the native API whose related virtualized API you want to reconfigure. This shows the details of the native API.
3. In the actions bar for the native API, click **Virtualize**.

This opens the **Virtualize <API_Name> (Step 1 of 3)** wizard.

- In the **Reconfigure an Existing Virtual Alias** field, select the virtualized API (also called as the Virtual Alias) you want to reconfigure. Complete the fields.

<u>In this field...</u>	<u>Do the following...</u>
Endpoint prefix for invocation alias	<p><i>Optional..</i> Enter a prefix for the alias. It accepts URL paths.</p> <p>For example, /testmethod/myprefix/, test@1234, and so on.</p>
Endpoints of <API_name> to be virtualized	<p>Select the endpoint you want to use for invoking the virtualized API.</p> <p>The Endpoints list displays the URLs of all endpoints that are available for the native API.</p>
Resources	<p>Select the resources you want to apply for invoking the virtualized API. For more information about using this field, see "Resource Synchronization in Virtual REST APIs" on page 214.</p>

- If you want to modify the run-time policy actions that are configured for the virtualized API, click **Next**.

In the **Virtualize <API_Name> (Step 2 of 3)** wizard, drag and drop the policy actions, and reconfigure the action parameters.

Then click the **Virtualize** button to save the changes.

If you want to make changes related to the gateways, follow the instructions in "[Publishing and Unpublishing APIs to and from Gateways](#)" on page 231.

Reconfiguring a Virtualized API from the Virtualized API Details Page

To reconfigure a virtualized API from its details page, proceed as follows:

- Display the list of virtualized APIs (see *Working with the CentraSite Business UI* for details).
- In the displayed list, click the link of the virtualized API you want to reconfigure. This shows the details of the virtualized API.
- In the actions bar for the virtualized API, click **Virtualize**.
- In the **Virtualize <API_Name> (Step 1 of 2)** wizard, drag and drop the policy actions, and reconfigure the action parameters.

Then click the **Virtualize** button to save the changes.

If you want to make changes related to the gateways, in the **Virtualize <API_Name> (Step 2 of 2)** wizard, follow the instructions in "[Publishing and Unpublishing APIs to and from Gateways](#)" on page 231.

Resource Synchronization in Virtual REST APIs

REST APIs are bound to change and evolve over time as they move through development, test and production. Modifications or enhancements such as, adding additional resources, HTTP methods or parameters, modifying the existing capabilities of resources, methods or parameters are performed on a particular REST API. In the case of a REST API that is virtualized to create a new Virtual REST API, keep in mind that the Virtual REST API which is newly created is an identical copy of the existing Native REST API and not just a reference REST API. This means that after a Virtual REST API is created, any subsequent changes in the Native REST API are not automatically propagated to the Virtual REST API. This means that if you add Resource B to the Native REST API, the newly added Resource B is not added to the existing Virtual REST API.

CentraSite provides the ability to deal with evolution of REST APIs. Depending on the nature of change in the Native REST API and your versioning strategy, CentraSite offers different mechanisms:

- **Versioning the REST APIs:** The CentraSite Business User Interface allows versioning of Native REST APIs and Virtual REST APIs. You can make the following kinds of versions on the REST APIs:
 - Create a new version of the existing Native REST API, make the necessary changes to the new version, and then create a Virtual REST API from the new version. We recommend that you use this kind of versioning when there is a requirement for considerable change in the Native REST API.
 - Create a new version of the existing Virtual REST API, make the necessary changes to the new version, and then republish the Virtual REST API to gateways.

For more information about versioning the REST APIs, see "[Versioning Virtual API or Service](#)" on page 219

- **Resynchronizing the REST APIs:** The CentraSite Business User Interface allows resynchronization of the Virtual REST APIs. This means that you copy the resource definition of the Native REST API (which includes HTTP methods, parameters, or sample requests and responses) to the Virtual REST API without affecting other configurations such as, the run-time policy actions and the consumption settings.

CentraSite Business UI offers the possibility of synchronizing the REST resource metadata for a Virtual REST API.

Note that there is no automatic synchronization of resource metadata between the REST API and the Virtual REST API. If you want to have the resource metadata in the Virtual REST API to be the same as the resource metadata in the REST API, you must manually

reconfigure the resource metadata in the Virtual REST API to synchronize with the resources in the REST API.

The user must have API Runtime Provider role to manage synchronization for the specified Virtual REST API.

When planning for the resource synchronization in Virtual REST API, as a best practice, take the following points into consideration:

- If you want to simply reconfigure the run-time policy action configurations for a particular Virtual REST API, we recommend that you use the **Virtualize** action in the details page of that Virtual REST API.
- If you want to reconfigure the alias field labeled **Endpoint prefix for invocation alias** or synchronize resources for a particular Virtual REST API, we then recommend that you use the **Virtualize** action in the details page of that Native REST API.
- Be aware that only the additions and deletions at the resource-level are recognized by the **Virtualize** action during reconfiguration of a Virtual REST API. Any changes that are made at the API-level and method-level are not recognized.

When you allow synchronization to happen on the Virtual REST API, the updated metadata of the Native REST API will completely overwrite the existing metadata of the Virtual REST API.

For example, if you have a Native REST API with two resources - Resource A and Resource B. Consider these two resources have the following methods:

Resource A	Resource B
GET Method	GET Method
POST Method	PUT Method

Assume you virtualize the Native REST API to create a Virtual REST API.

The Virtual REST API will include the two resources - Resource A and Resource B with the above specified methods.

Consider you update the details page of the Native REST API to include an additional metadata "*DELETE Method*" to the existing Resource B. The Native REST API will now have the two resources with the following methods:

Resource A	Resource B
GET Method	GET Method
POST Method	PUT Method
	<i>POST Method</i>

Consider you update the details page of the Virtual REST API to include an additional metadata “*DELETE Method*” to the Resource B. The Virtual REST API will now have the two resources with the following methods:

Resource A	Resource B
GET Method	GET Method
POST Method	PUT Method
	<i>DELETE Method</i>

When you reconfigure the Virtual REST API from the details page of the Native REST API, in the **Recreate resources** section, CentraSite displays both the resources - Resource A and Resource B (with a comment “already exists”) with pre-selected check boxes by default.

Note that the Resource B has a different set of method specification for the Native and Virtual REST APIs. However, on reconfiguring the Virtual REST API, this difference of the method specification is not recognized in the user interface.

Now when you choose to reconfigure the Virtual REST API with the default selection, upon synchronization, the existing metadata of the Virtual REST API will be completely overwritten by the updated metadata of the Native REST API. As a result, the Resource B of the Virtual REST API will now contain the inherited **POST Method**; but the user-defined **DELETE Method** will no longer exist.

Important: As a best practice, we recommend that you maintain the Native REST API as the single point of truth and synchronize all your changes with the Virtual REST API using the synchronization process.

Resource Synchronization Usage Scenarios

There are significant use cases that require real-time and accurate data synchronization. Consider a REST API allows users access to a collection of resources through a Virtual REST API. The goal is to keep the data on the Virtual REST API synchronized with the data of Native REST API. Types of use cases that are contemplated within the scope of the synchronization are exemplified by, but not limited to, the following scenarios:

- **Edit Resource Usage Scenario:** You choose to reconfigure an existing Virtual REST API, and the Native REST API indicates resources that are also available in the Virtual REST API. For an exemplary illustration, see ["Edit Resource Usage Scenario" on page 217](#).
- **Add Resource Usage Scenario:** You choose to reconfigure an existing Virtual REST API, and the Native REST API indicates resources that have been introduced since

the Virtual REST API was initially created. For an exemplary illustration, see ["Add Resource Usage Scenario" on page 218](#).

- **Delete Resource Usage Scenario:** You choose to reconfigure an existing Virtual REST API, and the Native REST API indicates resources that have been deleted since the Virtual REST API was initially created. For an exemplary illustration, see ["Delete Resource Usage Scenario" on page 218](#).
- **Combination Usage Scenario:** You choose to reconfigure an existing Virtual REST API, and the Native REST API indicates resource metadata that has undergone various changes (modifications, additions, deletions) since the Virtual REST API was initially created. For an exemplary illustration, see ["Combination of Usage Scenarios" on page 218](#).

The outcome of each operation (edit, add, and delete) on the resource metadata is given based on a very simple instance configuration in each of the usage scenarios.

Key	Description
Native REST API	An instance of the type "REST Service"
Virtual REST API	An instance of the type "Virtual REST Service"

Edit Resource Usage Scenario

Native REST API	Virtual REST API
Resource A	Resource A (already exists)
Resource B (updated)	Resource B (already exists)

In this scenario, CentraSite displays the Resource A and Resource B that are available in both the Native REST API and the Virtual REST API with pre-selected check boxes.

- To allow synchronization of the updated resource metadata (that is, include the modification to Resource B in the Virtual REST API), under the **Recreate resources** section, retain the default check box selection, and click **Next**.
- To ignore synchronization, under the **Recreate resources** section, unselect all of the check boxes, and click **Next**.

Add Resource Usage Scenario

Native REST API	Virtual REST API
Resource A	Resource A (already exists)
Resource B	Resource B (already exists)
Resource C(newly added)	Resource C (newly added)

In this scenario, CentraSite displays the Resource C which has been newly added to the Native REST API with a unselected check box.

- To allow synchronization of the newly added resource (that is, add the new Resource C to the Virtual REST API), under the **Recreate resources** section, select the check box of the newly added Resource C, and click **Next**.
- To ignore synchronization, under the **Recreate resources** section, unselect all of the check boxes, and click **Next**.

Delete Resource Usage Scenario

Native REST API	Virtual REST API
Resource A	Resource A (already exists)
Resource B (deleted)	Resource B (differences)

In this scenario, CentraSite displays the Resource B which has been deleted from the Native REST API with a unselected check box.

- To allow synchronization of the deleted resource metadata (that is, delete the Resource B from the Virtual REST API), under the **Resource differences** section, select the check box of the Resource B you want to delete, and click **Next**.
- To ignore synchronization, keep all of the check boxes unselected in the **Recreate resources** and **Resource differences** sections, and click **Next**.

Combination of Usage Scenarios

Native REST API	Virtual REST API
Resource A (deleted)	Resource A (differences)

Native REST API	Virtual REST API
Resource B (updated)	Resource B (already exists)
Resource C (added)	Resource C (newly added)

In this scenario, CentraSite displays the updated Resource B with a pre-selected check box, and the deleted Resource A and newly added Resource C with an unselected check box.

The resource synchronization mechanism can be best understood with the following table:

Use case	Recreate resources...			Resource differences...		
	Resource A	Resource B	Resource C	Resource A	Resource B	Resource C
Complete synchronization	N/A	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	N/A	N/A
Synchronize modified resource only	N/A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N/A	N/A
Synchronize added resource only	N/A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	N/A	N/A
Synchronize deleted resource only	N/A	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	N/A	N/A
No synchronization	N/A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N/A	N/A

Versioning Virtual API or Service

The CentraSite Business User Interface (BUI) allows versioning of a virtual API or service and publish a specified version of an API or service to Mediator.

The version can be system defined version, for example, 1.0, 2.0, or 3.0. Optionally, you can also define the user version of a versioned service, for example, u1, u2, or u3. Versioning allows you to modify the virtual service that is currently deployed in

CentraSite and publish the modified virtual service to Mediator. It also allows parallel coexistence of versions. You can select the version that you want to publish to Mediator.

Versioning of virtual API or services can be considered for the following scenarios:

- **Modifications to the definition of the service:** Services evolve over time. Modifications or enhancements such as, adding additional operations or methods, adding resources, or modifying the existing capabilities of a service are performed on a service. These modifications may affect the existing consumers or the consumers may not be able to adopt the changes for some time and thus the need for versioning of a service with parallel existence of old and new version arises.
- **Modifications to policies enforced for a particular service:** The policies enforced on a particular service can be changed or modified over a period of time. For example, in order to add new requirements of a new consumer, versioning can be useful to distinguish between the current and the new consumer requirements.

By default, for every version of a virtual service a system version number is generated. Optionally, you can also provide a version which is set as a user version.

Note: The user version (if available) takes precedence over the system version which is displayed in both CentraSite and Mediator. If the user version is not available, the system version is used.

Creating a Version for Virtual API or Service

A service can be versioned when:

- a virtual service (v1) is created either from scratch or from native service v1. The subsequent versions of this virtual service (v1) can be created by versioning v1.
- a virtual service (v1) is created from native service (v1) , on versioning, the native service (v2) is created. Virtual service (v2) is created by newly virtualizing native service (v2).

To create a new version for an existing service:

1. In the CentraSite BUI, go to the API or service details page.
2. In the action bar of the API or service, click **Version**.
3. In the **Add a New Version** wizard, specify the following fields:

In this field...	Specify...
Namespace	<i>Optional.</i> A namespace for the new versioned virtual service. By default, the URL of the service to be versioned is displayed.
Version	<i>Optional.</i> A version identifier for the virtual service. The version identifier is an optional attribute that you can

In this field...	Specify...
	<p>increment when a service is modified to indicate that the service has been updated. It can contain alphanumeric characters and special characters.</p> <p>If no value is specified, a system generated number is displayed.</p>
Organization	Specify the organization to which this versioned virtual service will be added.
Change Log	<i>Optional.</i> To add a comment for the versioned virtual API or service in order to keep track of the modifications made (if any).
Version dependent assets	<i>Optional.</i> Select this checkbox if you want to version the assets dependent on the service that is versioned.

Note: Only the newly versioned API or service can be versioned. For example, if a virtual service V1 having version as 1.0 is versioned to 2.0, virtual service 1.0 cannot be versioned, only the virtual service versioned to 2.0 can be versioned.

4. Click **Version**.

A new version of the service or the API is created in CentraSite with the version details specified. A copy of the current virtual service which consists of all the details, references, consumers, and local policy definitions is created and linked to the previous version of the service.

Note: The request can be routed to the oldest version of the deployed versions of the virtual service using the `pg.default.enable.oldVersion` property. If this property is set to true (the default) and a virtual service is executed without a version, then the request is routed to the oldest version of the deployed versions of the virtual service. If this property is set to false and a virtual service is executed without a version, then the request is routed to the newest version of the deployed versions of the virtual service.

For more information on versioning, refer to the **Versioning Support in Mediator** section in the *Administering webMethods Mediator* guide.

Publishing a Versioned Virtual API or Service

1. In the CentraSite BUI, go to the API or service details page.
2. Select the version of the service or the API that you want to publish to Mediator.

3. In the action bar of the API or service, click **Publish**. The **Publish** dialog box is displayed.
4. In the **All Gateway** drop-down box, select the gateway to which you want to publish the API or the service.
5. Click **Publish** to publish the versioned virtual API or service to Mediator.

Runtime Versioning Support in Mediator

Mediator supports deployment of versioned virtual APIs or services. You can deploy one or more versions of a virtual API or service to Mediator. The deployed versions of virtual API or service can be used simultaneously. You can provide a unique version to a particular virtual service or an API in CentraSite and publish a specified version of an API or a service to Mediator.

Each version is a unique service and the deployed versions are displayed in the Services page under the CentraSite navigation panel. Each version is unique and you must use an appropriate WSDL to generate the requests.

On creating a new virtual alias in Mediator, you can enter an endpoint prefix for the alias to be invoked in the **Endpoint prefix for invocation alias** field. The endpoint prefix provided in this field is displayed in the **URL** field of the VSD .

The information entered in the **Endpoint prefix for invocation alias** field is also available in the WSDL file.

Note: While generating the request, the content in the requests can be different for different versions. Hence, ensure that you select an appropriate WSDL of the version used to generate the request.

Sample URLs for Endpoints with Versioning

The version can be system defined version, for example, 1.0, 2.0, or 3.0. Optionally, you can also define the user version of a versioned service, for example, u1, u2, or u3. These system or user defined versions are displayed in the URLs for endpoints as follows:

- If the system defined version is 1.0 and if no user version is defined, then this system defined version is appended after the service name in the URL. For example,

```
http://127.0.0.1:5555/ws/<service name>/<system defined version>
http://127.0.0.1:5555/ws/TestVS1/1.0
```

- If you have defined the version of a versioned service, for example, ver2, then the version defined by you is appended after the service name in the URL. For example,

```
http://127.0.0.1:5555/ws/<service name>/<user defined version>
http://127.0.0.1:5555/ws/TestVS1/ver2
```

- If you have provided a prefix for the alias in the **Endpoint prefix for invocation alias** field, for example, /testing/EPprefix, then this endpoint prefix for invocation alias is appended before the service name in the URL. For example,

```
http://127.0.0.1:5555/ws/<endpoint prefix for invocation alias>/<service name>/
```

```
<system or user defined version>  
http://127.0.0.1:5555/ws/testing/EPprefix/TestVS1/ver2
```

Viewing or Changing a Virtualized API

This section describes how to view the information stored for a virtualized API and how to change them.

When editing the details of a virtualized API, keep the following points in mind:

- If you are not the owner of a particular virtualized API, you cannot edit that virtualized API unless you have a Modify permission on the API (granted through either a role-based permission or an instance-level permission).
- When you view the details for the virtualized API, you will only see profiles for which you have View permission. You will only be able to edit the profiles on which you have Modify permission.
- In addition to the above described permissions for managing a particular virtualized API, make sure that you have the specific roles and permissions required for reconfiguring the run-time actions and republishing the virtualized APIs to Mediator and API-Portal gateways as specified in ["Roles and Permissions Needed to Create and Manage Virtualized APIs" on page 188](#).
- Some attributes are designed to be read-only and cannot be edited even if they appear in a virtualized API on which you have Modify permission.
- Some attributes accept only specific types of information.
- Some attributes are designed to be read-only and cannot be edited even if they appear in a virtualized API on which you have Modify permission.

To view or modify the details that you have stored for a virtualized API, proceed as follows:

To view or modify the virtualized API's details

1. In CentraSite Business UI, display the list of virtualized APIs (see *Working with the CentraSite Business UI* for details).
2. In the displayed list, click the link of the virtualized API whose details you want to view or modify. This shows the details of the virtualized API.

The details include:

- The virtualized API's basic information (the virtualized API's version identifier, asset type, owning organization, last modified date, owning user, and a general description of the Native Virtual).

In addition to the generic attributes listed above, the virtualized API contains the following information, based on its consumption.

Attribute	Description
Native Service	Displays the Native API that is associated with the virtualized API.
Watchers	The number of users who are watching the virtualized API.
Consumers	The number of users who are registered as consumers for the virtualized API.
Consumed Assets	The number of assets to which the virtualized API is registered as a consumer.
Pending Approvals	The number of requests that are pending for approval.
Requested User	Displays the consumer who has requested for an API key or an OAuth2 Client access token.

- Additional information about the virtualized PI (namely, technical and specification details, provider and consumer information, gateways to which the virtualized API is published, and so on.)
3. If you want to make changes to the virtualized API's generic details that are displayed in the **Basic Information** section, click the **Edit** icon. You can then enter new values for the API's fields.
 4. If you want to make changes related to the extended attributes of the virtualized API, do the following:
 - a. Select the profile that contains the attribute(s) that you want to modify.
 - b. Edit the attributes on the profile as necessary.
 - c. Repeat steps 3.a and 3.b for each profile that you want to edit.
 5. Then click the **Save** icon to save the changes.

If you want to make changes related to the virtualized API's run-time configuration, follow the instructions in ["Assigning Run-Time Actions to a Virtualized API" on page 228](#).

If you want to make changes related to publishing the virtualized API to the gateways, follow the instructions in ["Publishing and Unpublishing APIs to and from Gateways" on page 231](#).

Viewing the Virtualized API Specific Profiles

The details of each virtualized API include the following additional information:

- Identification Profile
- API Key Identification Profile
- API Key Scope Profile
- OAuth2 Identification Profile

Identification Profile

In this profile, you specify the precise values for the consumer identifier token(s) that you want to use for identifying and authorizing the consumers for a particular virtualized API. (Alternatively, you may configure this profile to allow unrestricted access.)

For example, if you configure the Identification profile to identify and authorize consumers by IP address, the PEP extracts the IP address from a request's HTTP header at run time and searches its list of consumers for the virtualized API that is defined by that IP address.

- Note:**
- If you want to authenticate consumers, make sure that your policy enforcement point is configured to enable authentication. For information, see *Administering webMethods Mediator*.
 - For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

Field	Description
IPv4 Address	<p>Use this field to identify consumers based on their originating 4-byte IP address range.</p> <p>Specify a range of IPv4 addresses. Type the lowest IP address in the From field and the highest IP address in the To field. For example, 192.168.0.0 and 192.168.0.10</p> <p>The virtualized API will then identify and authorize only those requests that originate from the specified IP address.</p> <p>If you need to specify additional IP addresses, use the plus button to add more rows.</p>
IPv6 Address	<p>Use this field to identify consumers based on their originating 128-bit IPv6 address.</p> <p>Specify a IPv6 address. For example, <code>fd5a:5cc1:23:4::1f</code></p> <p>The virtualized API will then identify and authorize only those requests that originate from an IP address that lies between the specified ranges.</p>

Field	Description
Hostname	<p>If you need to specify additional IP addresses, use the plus button to add more rows.</p> <p>Use this field to identify consumers based on a specified host name.</p> <p>Specify the hostname. For example, <code>pcmachine.ab.com</code></p> <p>The virtualized API will then identify and authorize only those requests that originate from the specified host name.</p> <p>If you need to specify additional host names, use the plus button to add more rows.</p>
HTTP Token	<p>Use this field to authenticate consumers based on the user name that is transmitted in an HTTP authentication user token.</p> <p>Specify one or more HTTP user names. For example, <code>SAGUser123</code></p> <p>The virtualized API will then identify and authorize only those requests that contain the specified user name encoded and passed in the HTTP authentication user token.</p> <p>If you need to specify additional tokens, use the plus button to add more rows.</p>
WS-Security Token	<p>Use this field to authenticate consumers based on the user name that is transmitted in the SOAP or XML message header (HTTP body).</p> <p>Specify the WSS username token. For example, <code>userwss</code></p> <p>The virtualized API will then identify and authorize only those requests that contain the specified user name passed in the SOAP or XML message header.</p> <p>If you need to specify additional tokens, use the plus button to add more rows.</p>
XPath Token	<p>Use this field to identify consumers based on the result of applying an XPath expression on the SOAP or XML message or request.</p> <pre data-bbox="516 1745 1333 1822"> //*[local-name()='Envelope']/* [local-name()='Body']/* [local-name()='echoInt']/* [local-name()='echoIntInput='] [.='2'] </pre> <p>The virtualized API will then identify and authorize only those requests that contain the XPath and the consumers.</p>

Field	Description
	If you need to specify additional tokens, use the plus button to add more rows.
Consumer Certificate	<p>Use this field to identify consumers based on information in an X.509 v3 certificate.</p> <p>Click Upload to locate and select the certificate (.cer) file.</p> <p>The virtualized API will then identify and authorize only those requests that contain the specified X.509 v3 certificate in the SOAP or XML header.</p>

Identification Profile (for Assets with Key-based Authentication)

Field	Description
API Key String	<p><i>Read-only.String.</i> The confidential secret key used to securely authenticate the consumer.</p> <p>The API Key String field is visible only to the consumer who requested an API key.</p>
Expiry Date	<i>Read-only.String.</i> An expiration date for the API key.

OAuth2 Identification Details Profile (for Assets with OAuth-based Authentication)

Field	Description
Client Id	<i>Read-only.String.</i> The unique identifier that is used by the client to fetch access tokens for the virtualized API.
Client Secret	<i>Read-only.String.</i> The secret key value that is used with the client identifier, serves as a password to fetch access tokens for the virtualized API.
Client Name	<i>Read-only.String.</i> The name of the client (consumer application) that is attempting to get access to the virtualized API.
Scope	<i>Read-only.String.</i> The scope value is the name of the virtualized API. If the scope value is valid, Mediator obtains the access token. If no scope value is provided, Mediator

Field	Description
	provides the access token to the scope in which the client is allowed and adds the scope to the response.
Refresh Token	<i>Read-only.String.</i> The unique identifier used by the client to obtain a new access token when the current access token becomes invalid or expires.

API Key Scope Profile

Field	Description
API Service	<i>Read-only.String.</i> The name of the virtualized API that is associated with the API key. To view details of the virtualized API, click its hyperlinked name.

Assigning Run-Time Actions to a Virtualized API

This section describes how to use the **Virtualize** action to configure the policy actions for the virtualized API.

To access the **Virtualize** action, you must have the following roles or permissions:

- CentraSite Administrator
- Organization Administrator
- Asset Provider
- Instance-level Modify permission on the native API or virtualized API

In addition to the above described roles and permissions for accessing the **Virtualize** action in the details page of a particular virtualized API, make sure that you have the specific API Runtime Provider role for reconfiguring the run-time actions of the virtualized APIs as specified in ["Roles and Permissions Needed to Create and Manage Virtualized APIs" on page 188](#).

For details about roles and permissions, see *Getting Started with CentraSite*.

Before You Begin

The **Virtualize <API_Name> (Step 2 of 3)** wizard specifies the list of actions to govern the run-time behavior for the virtualized API.

When you define the actions for a virtualized API, keep the following points in mind:

- A run-time policy configuration is valid if:

- it consists of at least one action in each of these stages - **Receive** and **Routing**.
- there is a valid endpoint configured in the **Route to** field of the Routing action.
- When you drag an action from the **Policy Actions** area, the respective step in the **Message Flow** area highlights where the action fits in, thus making the navigation from **Policy Actions** area to the **Message Flow** area more intuitive.
- Not all stages support the full set of actions. Every action happens only within a respective step. For example, the “Evaluate” actions occur only on the **Enforce** stage; while the “Routing” actions occur only on the **Routing** stage.
- Mediator executes the policy actions configured for the virtualized API in a predefined order. For information about how Mediator executes actions (and how to avoid policy conflicts), see ["Effective Policies" on page 390](#).
- Some actions are mutually dependent. That is, a specific action must be used in conjunction with another particular action. For example, a **Message Flow** area that includes the [Set JMS Headers](#) action must also include the [JMS Routing Rule](#) action.
- Some actions are mutually exclusive. That is, a specific action cannot be used in conjunction with another particular action. For example, a **Message Flow** area that includes the [JMS Routing Rule](#) action cannot include the [Straight Through Routing](#) action.
- Some of the actions are allowed to appear multiple times within a message flow step. For those actions that can appear in a message flow only once (for example, Evaluate IP Address), Mediator will choose only one, which might cause problems or unintended results.
- You can view a tooltip text for any accordion by moving the cursor over the accordion name. The tooltip text gives a summary of the accordion’s purpose.
- If you modify the policy action for a virtualized API which is already published to a Mediator gateway, CentraSite automatically republishes the modified virtualized API.
- If you want to enable the REST support for a Virtual SOAP-based API, ensure that the **Enable REST Support** action is included in the **Receive** stage for the API. For more information, see ["Exposing a Virtual SOAP API as Virtual REST API" on page 244](#)
If you include the **Enable REST Support** policy action in a SOAP API configuration, clients who can only send REST requests can now invoke a REST-enabled SOAP API using both a SOAP request and a REST request in Mediator, and using a REST request in API-Portal.
- If you are using Mediator as your gateway, you must include at least one Evaluate * action in order to identify or validate the consumers. For common usage cases of identification and validation actions, see *Run-Time Governance with CentraSite*.
For information about the individual run-time actions that are supported out-of-the-box, see ["Built-In Run-Time Actions Reference for APIs" on page 382](#).

- Be aware that actions from the WS-I category cannot be combined with other types of actions. Also be aware that when you add a WS-I action to the action list, CentraSite will automatically add dependent actions to the list as necessary.

Ways in Which You Assign Run-Time Actions to Virtualized APIs

You can assign run-time actions to an existing virtualized API in the following two ways:

- Using the **Virtualize** action in the details page of the native API.
- Using the **Virtualize** action in the details page of the virtualized API.

Modifying the Action List

Use the following procedure to modify the action list for a virtualized API.

To modify the action list for a virtualized API

1. Display the details page for the virtualized API whose action list you want to modify. If you need procedures for this step, see "[Viewing or Changing a Virtualized API](#)" on page 223.
2. In the details page of the virtualized API, click the **Virtualize** icon.
This opens the **Virtualize <API_Name> (Step 1 of 2)** wizard.
3. To add an action to the **Message Flow** area, proceed as follows:
 - a. In the **Policy Actions** area, expand the desired accordion (Request Handling, Policy Enforcement, Response Handling or Error Handling).
 - b. Locate the action you want to add for the virtualized API.
 - c. Drag and drop the action in the appropriate stage (Receive, Enforce, Routing, or Response) in **Message Flow** area.
 - d. Repeat the above steps for each action that you want to add.
4. To configure the parameter values for any new actions that you might have added to the policy action list, or to make any necessary updates to the existing action parameter values, follow the instructions as given below in "[Configuring Policy Action Parameters](#)" on page 231.
5. To remove an existing action from the **Message Flow** area, proceed as follows:
 - a. Mouse hover the action you want to remove. This causes an icon to appear, that you can use for removing.
 - b. Click the icon to remove the action
 - c. Repeat the above steps for each action that you want to remove.

- When the action list is complete and you have configured all of the input parameters for the actions correctly, click the **Virtualize** button to save the changes.

Configuring Policy Action Parameters

Policy actions have parameters that you must set to configure the behavior of the action at enforcement time.

To configure the action parameters

- Display the details page for the virtualized API whose policy action parameters you want to configure. If you need procedures for this step, see ["Viewing or Changing a Virtualized API" on page 223](#).
- In the details page of the virtualized API, click the **Virtualize** icon.
This opens the **Virtualize <API_Name> (Step 1 of 2)** wizard.
- To configure the required parameters for the policy actions displayed in the **Message Flow** area, proceed as follows:
 - Mouse hover the action whose parameters you want to modify.
 - Choose the **Configure** icon to the right of the action name.
 - In the **<action_name>** dialog box, set the values for the parameters as necessary.
For information about the parameter settings for the built-in run-time actions provided by CentraSite, see ["Run-Time Actions Reference" on page 402](#).
 - Click **OK** to save the parameter settings.

Note: If you fail to specify the required parameters, the system alerts you with a red error icon. Pointing to the error icon shows a hint with the error description.

- Repeat the above steps for each action that you want to modify.

Important: If you make changes to the API's run-time enforcement, for example, **Enable REST Support** action for a SOAP API, you must manually republish the virtualized API to put those changes into effect.

- When you have configured all of the required parameters for the virtualized API, click the **Virtualize** button to save the changes.

Publishing and Unpublishing APIs to and from Gateways

API management solution allows you to publish and unpublish APIs to and from webMethods Mediator and webMethods API-Portal gateways.

The Process of Publishing an API to Mediator

Publishing an API to Mediator gateway refers to the process you use to deploy virtualized APIs to Mediator where they are exposed to consuming clients.

The process of publishing an API to the Mediator gateway is carried out by a synchronous mechanism between CentraSite and Mediator.

Doing this involves the following high-level steps:

- **Step 1:** A user initiates the publish process by selecting the virtualized APIs that are to be published, and specifies in which Mediators they are to be published.
- **Step 2:** CentraSite publishes the virtualized APIs to each of the specified Mediators.
- **Step 3:** The publishing process continues even if CentraSite encounters a failure with one of the Mediator.
- **Step 4:** CentraSite logs information about the APIs that fail at the end of the publish process. If the process of publishing returned specific data about the API, the API's metadata is updated as needed in the CentraSite registry/repository.

The Process of Publishing an API to API-Portal

Publishing an API to API-Portal gateway refers to the process you use to deploy native and virtualized APIs to API-Portal on which they are to be exposed for testing and user consumption.

The process of publishing an API to the API-Portal gateway is initiated from CentraSite and is carried out on the API-Portal server.

Doing this involves the following high-level steps:

- **Step 1:** A user initiates the publish process by selecting the virtualized APIs that are to be published, and specifies in which API-Portals they are to be published.
- **Step 2:** CentraSite publishes the virtualized APIs to each of the specified API-Portals.
- **Step 3:** The publishing process continues even if CentraSite encounters a failure with one of the API-Portal.
- **Step 4:** CentraSite logs information about the APIs that fail at the end of the publish process. If the process of publishing returned specific data about the API, the API's metadata is updated as needed in the API-Portal registry.

Roles and Permissions Needed to Publish and Unpublish APIs

To publish and unpublish APIs to and from a Mediator gateway, you must have the following roles or permissions:

- CentraSite Administrator

- Organization Administrator
- Mediator Publisher
- Instance level Modify permission for Mediator gateway

To publish and unpublish APIs to and from an API-Portal gateway, you must have the following roles or permissions:

- CentraSite Administrator
- Organization Administrator
- API-Portal Publisher
- Instance-level Modify permission for API-Portal gateway
- If you have the CentraSite Administrator role, you can publish and unpublish APIs to and from any gateways within any organization.
- If you have the Organization Administrator role, Mediator Publisher role, or API-Portal Publisher role for a specific organization, you have the ability to publish and unpublish APIs to and from the Mediator or API-Portal gateways within that specific organization.

For more information about roles and permissions, see *Getting Started with CentraSite*.

Important Considerations When Publishing an API

Following are some things you should consider when you publish an API to the gateways.

- You will only be able to publish APIs to the gateways for which you have the required roles or the Modify instance-level permission.
- You can publish a native API to an instance of API-Portal. However, you cannot publish a native API to an instance of Mediator.
- You can publish virtualized APIs to both Mediator and API-Portal gateways.
- When publishing a virtualized API to both Mediator and API-Portal gateways in a single step, only if publishing to at least one Mediator instance was successful, CentraSite will publish the API to API-Portal instance.
- When publishing an API to one or more gateways, if CentraSite encounters a publish failure with one of the gateway, it immediately ignores the failure gateway and proceeds with the next gateway. Any gateway that encountered failure during the publish process is displayed in the Publish Log.
- When trying to publish an API to a set of gateways, if the API is already published to a selected gateway, then that API is republished to that particular gateway.

Use Cases for Publishing an API

When publishing an API to the Mediator and API-Portal gateways, consider the following use cases:

Use Case A: Publish an API From the Native API Details Page

In this use case, you use the details page of a native API for publishing to the Mediator and API-Portal gateways.

Publish an API Without a Virtualized API

Here, you publish an instance of native API (which does not currently have a virtualized instance) from its details page.

The following table summarizes how an instance of native service A (without a virtualized service A1) is applicable for publishing to both gateways in a single step.

In this use case, there is one principal scenario:

Scenario	Instance of API	Publish to...	
		Mediator	API-Portal
Native Service A without Virtualized Service A1	Native Service		✓

Publish an API With a Virtualized API

Here, you publish an instance of virtualized API (derived from the virtualization action of native API) from the details page of native API.

The following table summarizes how an instance of native service A (with a virtualized service A1) is applicable for publishing to both gateways in a single step.

In this use case, there is one principal scenario:

Scenario	Instances of API	Publish to...	
		Mediator	API-Portal
Native Service A with Virtualized Service A1	Native Service A Virtualized Service A1	✓	✓*

* Only if publishing of an API to at least one Mediator gateway has executed successfully.

For instructions on how to publish an API from the details page of a native API, see the section ["Publishing an Individual API from the Native API Details Page"](#) on page 239.

Use Case B: Publish an API From the Virtualized API Details Page

In this use case, you use the details page of a virtualized API for publishing to the Mediator and API-Portal gateways.

The following table summarizes how an instance of virtualized service B is applicable for publishing to both gateways in a single step.

In this use case, there is one principal scenario:

Scenario	Instance of API	Publish to...	
		Mediator	API-Portal
Virtualized Service B)	Virtualized Service B	✓	✓*

* Only if publishing of an API to at least one Mediator gateway has executed successfully.

For instructions on how to publish an API from the details page of a virtualized API, see the section ["Publishing an Individual API from the Virtualized API Details Page"](#) on page 241.

Use Case C: Publish an API from the Search Results Page

In this use case, you use the Search Results page to publish at set of native and virtualized APIs to the Mediator and API-Portal gateways.

The following table summarizes how a combination of the instances of native service A (with a virtualized service A1), and a virtualized service B are applicable for publishing to both gateways in a single step.

In this use case, there are six principal scenarios:

Scenario 1

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Native Service A	Native Service A		✓

Without

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Virtualized Service A1			
Virtualized Service B (not published to Mediator)	Virtualized Service B	✓	✓*

Scenario 2

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Native Service A	Native Service A		✓
<i>Without</i> Virtualized Service A1			
Virtualized Service B (already published to Mediator)	Virtualized Service B	✓	✓*

Scenario 3

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Native Service A	Native Service A		
<i>With</i> Virtualized Service A1 (not published to Mediator)	Virtualized Service A1	✓	✓*

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Virtualized Service B (not published to Mediator)	Virtualized Service B	✓	✓*

Scenario 4

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Native Service A	Native Service A		
<i>With</i> Virtualized Service A1 (not published to Mediator)	Virtualized Service A1	✓	✓*
Virtualized Service B (already published to Mediator)	Virtualized Service B	✓	✓*

Scenario 5

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Native Service A	Native Service A		
<i>With</i> Virtualized Service A1 (already published to Mediator)	Virtualized Service A1	✓	✓*

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Virtualized Service B (not published to Mediator)	Virtualized Service B	✓	✓*

Scenario 6

Scenarios	Instances of API	Publish/Unpublish	
		Mediator	API-Portal
Native Service A	Native Service A		
<i>With</i> Virtualized Service A1 (already published to Mediator)	Virtualized Service A1	✓	✓*
Virtualized Service B (already published to Mediator)	Virtualized Service B	✓	✓*

* Only if publishing of an API to at least one Mediator gateway has executed successfully.

For instructions on how to publish a set of APIs from the Search Results page, see the section "[Publishing Multiple APIs from the Search Results Page](#)" on page 242.

Publishing API(s) to Gateway(s)

The following procedures describe how to publish a single API or a set of APIs to the gateways.

Publishing an Individual API from the Native API Details Page

When you execute a publish action from the native API details page, the exact rendering of user interface depends on whether or not the native API has an immediate virtualized API.

- If the native API does not have an immediate virtualized API, you will be allowed to directly publish the native API to API-Portal gateway.
- If the native API has an immediate virtualized API, you will be directed to choose the virtualized APIs, and then publish them to both gateways.

Perform these steps to publish an individual API from the details page of a native API.

To publish a single API using details page of native API

1. In CentraSite Business UI, display a list of available APIs.
2. In the displayed list, click the name of the native API.
3. In the action bar of the native API, click **Publish**.
4. If the displayed native API does not have an immediate virtualized API, CentraSite directs you to the **Publish to Gateway** dialog box.
5. Moreover, if the displayed native API has a virtualized API, you will have an intermediate **Publish Virtual APIs** dialog box. This dialog box allows you to configure the virtualized API for publishing to the gateways.

The **Publish Virtual APIs** dialog box lists the names of all of its virtualized APIs.

- a. Mark the checkbox of each virtualized API you want to publish to the gateways.
 - b. Click **Next**. This opens the **Publish to Gateway** dialog box.
6. In the **Gateway** drop-down box, choose the gateway to that you want to publish the API.

The gateways displayed in the drop-down box are the gateways for that you have the Modify permission.

The gateways list includes the following:

If you choose...	CentraSite will display...
Mediator	Instances of the Mediator gateway.
API-Portal	Instances of the API-Portal gateway.
All Gateways	All instances of both the Mediator and API-Portal gateways

By default, the **Gateway** drop-down box is set to **All Gateways**.

7. If you want to further restrict the list of gateways, you can filter the gateways based on the sandbox by which they were classified.

When you filter gateways using a sandbox, CentraSite displays gateways whose endpoints are classified by the specified sandbox. For example, you could use a **Production** sandbox to filter the gateways whose endpoints were classified using the **Production** sandbox.

In the **Sandbox** drop-down box, choose the category by which the gateways were classified.

Note that apart from the predefined categories, you can also use your custom categories to filter the gateways. The predefined categories include:

If you choose...	CentraSite will display...
Development	Instances of a specified gateway whose endpoints were classified by the Development category.
Production	Instances of a specified gateway whose endpoints were classified by the Production category.
Test	Instances of a specified gateway whose endpoints were classified by the Test category.
All Sandboxes	Instances of a specified gateway whose endpoints were classified by any of the above mentioned categories.

By default, the **Sandbox** drop-down box is set to **All Sandboxes**.

8. The gateways list provides the following information about each gateway:

Column	Description
Name	The name assigned to the gateway.
Type	The type of gateway.
Sandbox	The category that classifies deployment endpoint of the gateway.
Settings	The Configure icon displays the API-Portal Settings dialog box that enables you to configure a set of

Column	Description
	<p>sandbox categories for the specified API-Portal individually.</p> <p>The dialog box also lists the set of API communities available for the specified API-Portal. You can assign an API to one or more API communities.</p>

9. Mark the checkbox of each gateway to which you want to publish the API.
10. *Optional. Applicable for an API-Portal.* Select the sandbox category that classifies the endpoint of the API for publishing to API-Portal, and assign the API to one or more communities available for the specified API-Portal. Do the following:
 - a. Click the **Configure** icon next to **Sandbox** field.
This icon displays the **API-Portal Settings** dialog box.
 - b. Select the sandbox categories.
 - c. Select one or more communities from the list. By default, the API is assigned to the Public Community. Any new communities assigned will overwrite the existing assignments.
 - d. Click **OK**.
For more information on API communities, see *webMethods API-Portal Administrator's Guide*.
11. *Optional. Applicable for a Mediator.* Enable the **Expose to Consumers** option to allow guest users to simply access and examine the usage of API through CentraSite.
12. When you have finished making your settings, click **Publish**.
13. A **Publish Inprogress** popup will display the progress state of publishing the API to selected gateways.
If the publish process logs failures, identify and correct the failure and then try publishing the API again.

Publishing an Individual API from the Virtualized API Details Page

When you execute a publish action from the virtualized API details page, you will be allowed to directly publish the virtualized API to both gateways

Perform these steps to publish an individual API from the details page of a virtualized API.

To publish a single API using details page of virtualized API

1. In CentraSite Business UI, display a list of available APIs.

2. In the displayed list, click the name of the virtualized API you want to publish to the gateways.
3. In the action bar of the virtualized API, click **Publish**. This opens the **Publish to Gateway** dialog box.
4. Repeat steps 6 through 13 as above for publishing the virtualized API to gateways as you need.

Publishing Multiple APIs from the Search Results Page

When you execute a publish action from the Search Results page, you will be allowed to directly publish multiple instances of native and virtualized APIs to both gateways in a single step.

Perform these steps to publish multiple APIs from the Search Results page.

To publish a set of APIs using Search Results page

1. In CentraSite Business UI, display a list of available APIs.
2. In the displayed list, mark the checkbox of each API you want to publish to the gateways.
3. In the action bar of the Search Result page, click **Publish**. This opens the **Publish to Gateway** dialog box.
4. Repeat steps 6 through 13 as above for publishing APIs to gateways as you need.

Unpublishing API(s) from Gateway(s)

The following procedures describe how to unpublish a single API or a set of APIs from both gateways.

Unpublishing an Individual API from the Native API Details Page

When you execute an unpublish action from the native API details page, the exact rendering of user interface depends on whether or not the native API has an immediate virtualized API.

- If the native API does not have an immediate virtualized API, you will be allowed to directly unpublish the native API from API-Portal gateway.
- If the native API has an immediate virtualized API, you will be directed to choose the virtualized APIs, and then unpublish them from both gateways.

Perform these steps to unpublish an API from the details page of a native API.

To unpublish a single API using details page of native API

1. In CentraSite Business UI, display a list of available APIs.

2. In the displayed list, click the name of the native API you want to unpublish from the gateways.
3. In the action bar of the native API, click **Unpublish**.
4. If the displayed native API does not have an immediate virtualized API, CentraSite directs you to the **Unpublish from Gateway** dialog box.
5. On the other hand, if the displayed native API has an immediate virtualized API, CentraSite directs you to an intermediate **Unpublish Virtual APIs** dialog box. This dialog box allows you to configure the virtualized API for unpublishing from the gateways.

The **Unpublish Virtual APIs** dialog box lists the names of its virtualized APIs which are already published to the gateways.

- a. Mark the checkbox of each virtualized API you want to unpublish from the gateways.
- b. Click **Next**. This opens the **Unpublish from Gateway** dialog box.
6. In the **Gateway** drop-down box, choose the gateway from that you want to unpublish the API.
7. If you want to further restrict the list of gateways, in the **Sandbox** drop-down box, choose the category by which the gateways were classified.
8. Mark the checkbox of each gateway from that you want to unpublish the API.
9. *Optional. Applicable for a Mediator.* Select the **Revoke Consumability** option to revoke access of API from guest users.
10. When you have finished making your settings, click **Unpublish**.
11. An **Unpublish Inprogress** popup will display the progress state of unpublishing the API from selected gateways.

If the unpublish process logs failures, identify and correct the failure, and then try unpublishing the API again.

Unpublishing an Individual API from the Virtualized API Details Page

When you execute an unpublish action from the virtualized API details page, you will be allowed to directly unpublish the virtualized API from both gateways

To unpublish a single API using details page of virtualized API

1. In CentraSite Business UI, display a list of available APIs.
2. In the displayed list, click the name of the virtualized API you want to publish from the gateways.
3. In the action bar of the virtualized API, click **Unpublish**. This opens the **Unpublish from Gateway** dialog box.

4. Repeat steps 6 through 11 as above for unpublishing the virtualized API from gateways as you need.

Unpublishing Multiple APIs from the Search Results Page

When you execute an unpublish action from the Search Results page, you will be allowed to directly unpublish multiple instances of native and virtualized APIs from both gateways in a single step.

To unpublish a set of APIs using Search Results page

1. In CentraSite Business UI, display a list of available APIs.
2. In the displayed list, mark the checkbox of each API you want to unpublish from the gateways.
3. In the action bar of the Search Result page, click **Unpublish**. This opens the **Unpublish from Gateway** dialog box.
4. Repeat steps 6 through 11 as above for unpublishing APIs from gateways as you need.

Exposing a Virtual SOAP API as Virtual REST API

CentraSite offers you the possibility to expose Virtual SOAP APIs also as Virtual REST APIs in webMethods Mediator.

The REST-enabled support for a Virtual SOAP API is achieved using the **Enable REST Support** policy action that comes pre-shipped with CentraSite.

When you include the **Enable REST Support** action in a Virtual SOAP API's run-time configuration, clients can invoke the REST-enabled Virtual API in Mediator using both a SOAP request and a REST request. For more details about the **Enable REST Support** action, see "[Built-In Run-Time Actions Reference for APIs](#)" on page 382.

This policy action is set by default in the **Receive** stage for all Virtual APIs. To disable the REST support for a Virtual API, manually remove the **Enable REST Support** action from the **Receive** stage of the Virtual API.

REST-Enabled Virtual SOAP API in webMethods API-Portal

When you publish a REST-enabled Virtual API to webMethods API-Portal, it is exposed only as a Virtual REST API in API-Portal. All the operations of the Virtual API are exposed as REST Resources. This allows clients to test the exposed Virtual REST API in API-Portal using a REST request.

Currently, API-Portal only supports the HTTP GET and POST methods with these kinds of REST Resources.

Limitations:

- If a SOAP operation exposed as a REST resource is invoked using a GET request, the required values for the SOAP operation should be sent as Query parameters. But these Query parameters are not pre-populated with the exposed REST resource. You must manually specify them.
- It is assumed that the default content-type is `application/json`.

Displaying Runtime Information for a Virtualized API

You can view the events and metrics for a virtualized API in the **Runtime Metrics** and **Runtime Events** profiles.

Prerequisites:

- You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For procedures, see *Administering webMethods Mediator*.
- You must configure CentraSite to receive run-time events from Mediator, as described in ["Run-Time Gateways" on page 59](#).
- You must have a View permission on the **Runtime Metrics** and/or **Runtime Events** profiles of the virtualized API.
- To view the run-time information for a virtualized API, it is necessary that the API's type's definition includes the **Runtime Metrics** and/or **Runtime Events** profiles.

The following sections describe how to view the run-time information of a virtualized API.

The Runtime Metrics

The **Runtime Metrics** profile includes the following types of Key Performance Indicator (KPI) metrics for a virtualized API. The KPI metrics are created by the webMethods Mediator.

Metric	Description
Availability	The percentage of time that a service was available during the current interval. A value of 100 indicates that the virtualized API was always available. If invocations fail due to policy violations, this parameter could still be as high as 100.
Success Count	The number of successful service invocations in the current interval.

Metric	Description
Total Request Count	The total number of requests (successful and unsuccessful) in the current interval.
Fault Count	The number of failed invocations in the current interval.
Average Response Time	The average amount of time it took the service to complete all invocations in the current interval. This is measured from the moment Mediator receives the request until the moment it returns the response to the caller.
Minimum Response Time	The minimum amount of time it took the service to complete an invocation in the current interval.
Maximum Response Time	The maximum amount of time it took the service to complete an invocation in the current interval.
Note:	By default, the Response Time metrics do not include metrics for failed invocations. To include metrics for failed invocations, set the <code>pg.PgMetricsFormatter.includeFaults</code> parameter to <code>true</code> . For more information about KPI metrics, intervals, and advanced settings, see <i>Administering webMethods Mediator</i> .

Displaying the Runtime Metrics

To display the run-time metrics for a virtualized API, proceed as follows:

To display run-time metrics for a virtualized API

1. Display the details page for the virtualized API whose run-time metrics you want to view.
2. In the details page of the virtualized API, select the **Runtime Metrics** profile.
3. Expand the **Filters** node.
4. Specify the exact set of attributes you want to use to filter the metrics list.

In this field...

Do the following...

Gateway

Select a particular gateway to which the virtualized API is published, or select **All** to view the metrics of all gateways to which the virtualized API is published.

In this field...	Do the following...
	CentraSite displays None by default.
Date Range	Specify a range of dates to view the metrics for the virtualized API. For example, Last 1 hour, Last 12 hours, Last 1 day, Last 5 days, Last 10 days, Last 20 days, Custom. CentraSite displays Last 10 days by default.
Start Date/End Date	If you have selected the Custom option in the previous field, specify the time period to view the metrics list. Start Date: Click the calendar and select a starting date and time. End Date: Click the calendar and select a ending date and time.
Display Interval	Specify the running count metrics of the virtualized API at the displayed time intervals. The interval is specified in the format 3m 2d 6h; wherein "m" indicates the month, "d" indicates the day and "h" indicates the hour.

5. Click **Refine**.

CentraSite displays a graphical view of the run-time metrics for all performance categories as shown below:

- **Multi-line Chart.** The chart shows the Minimum Response Time, Maximum Response Time, and Average Response Time of the API.
- **Pie Chart.** The chart shows the Success Request Counts, Total Request Counts, and Fault Counts of the API.
- **Gauge Chart.** The chart shows the availability of the API.

The Runtime Events

CentraSite can receive the following predefined run-time event types.

Event Type	Description
Lifecycle	A Lifecycle event occurs each time Mediator is started or shut down.
Error	An Error event occurs each time an invocation of an API results in an error.

Event Type	Description
Policy Violation	A Policy Violation event occurs each time an invocation of an API violates a run-time policy that was set for the API.
Transaction	A Transaction event occurs each time an API is invoked (successfully or unsuccessfully).
Monitoring	Mediator publishes key performance indicator (KPI) metrics, such as the average response time, fault count, and availability of all APIs (described below).
Note:	<p>Keep the following points in mind:</p> <ul style="list-style-type: none"> ■ For more information about run-time event types, see <i>Administering webMethods Mediator</i>. ■ For details about intervals, see <i>Administering webMethods Mediator</i>.

Displaying the Runtime Events

Use the following procedure to display run-time events for a virtualized asset.

To view the run-time events, the following prerequisites must be met:

- To view the run-time events of an API, it is necessary that the virtual type's definition includes the **Runtime Events** profile.
- If you do not see the **Runtime Events** profile of an API, it is probably because you do not have "View" permission for the profile.

To display run-time events for a virtualized API

1. Display the details page for the asset whose run-time events you want to view.
2. Select the **Runtime Events** profile.
3. Use the following fields to filter the event list you want to view:

In this field...	Specify...
Gateway	<p>A gateway to which the asset is deployed, or select All to view the event information of all gateways to which the API is deployed.</p> <p>CentraSite displays None by default.</p>
Consumer	<p>A consumer of the asset, or select All to view the run-time event information of all consumers of the asset.</p>

In this field...	Specify...
	CentraSite displays All by default. However, if you do not have at least one consumer registered in the registry, CentraSite displays None by default.
Event Type	A particular event type, or select All to view all event types. For a list of the supported event types, see " The Runtime Events " on page 247. CentraSite displays All by default.
Date Range	A range of dates from which to view the events (for example, Last 1 hour, Last 12 hours, Last 1 day, Last 5 days, Last 10 days, Last 20 days, Last 1 month, Custom, and so on). CentraSite displays Last 1 month by default.
Start Date/End Date	If chosen Custom in the previous field, then the time period for which to view the metrics. Start Date: Click the calendar and select a starting date and time. End Date: Click the calendar and select a ending date and time.
Display Interval	A running count events of the service displayed at regular time intervals. The interval is specified in the format 3m 2d 6h; wherein m indicates the month, d indicates the day and h indicates the hour.

4. Click **Refine**.
5. Expand the **Graphical** node to display a graphical view of the run-time event information.
6. Expand the **Tabular** node.

CentraSite displays a tabular view of the event information in the left pane.

Field	Description
Date/Time	The date/time that the event occurred. Click this hyperlinked value to view the Event Detail page, which will contain the event's SOAP request or response name in the Attribute column. Click the hyperlinked request or response name to display the full SOAP request or response.

Field	Description
Event Type	(Read-only.) The type of event (for example, Monitoring, Policy Violation, Error, and so on).
Gateway	(Read only.) The gateway on which the event occurred.

- To access the details of an event, click on the link for the event.

The **Event Details** dialog in the right pane shows a detailed information about the event that you select in the left pane.

Obtaining Your API Keys and Access Tokens for Consumption

The following section describes how to fetch your API keys and access tokens that enable you to consume APIs.

Note: To enable CentraSite to issue email messages, an administrator must first configure CentraSite's email server settings. For procedures, see the *CentraSite Administrator's Guide*.

Fetching and Using Your API Keys for Consumption

RESTful APIs are often exposed over the open Internet for consumption. API providers need a mechanism to prevent unauthorized access to the API. One approach is to provision consumers with API keys. Those keys can be used as authentication tokens.

CentraSite API Management Solutions automatically generates API keys when consumers request to consume APIs. The API providers can view, approve and set expiration for API keys. This ensures that no consumer can access a protected API without a valid key.

API keys are verified at runtime to ensure that:

- The API key presented is valid and has not expired.
- The API key presented is approved to consume an API that includes the URI in the request.

If you will be using the API key authentication and you have successfully registered as a consumer for an API, you should have received your API key details through an email message.

The API Consumption Model

To enable a consumer to consume an API, the following events must occur:

1. The consumer sends a request to consume a specified API. The request must include the consumer's authentication credentials.
2. CentraSite generates the API key for consumption of the API (the specific key generation steps depend on the configuration settings defined by the Provider of that particular API). Later, CentraSite prepares the API for publishing and invokes the API Key Generation policy on the Mediator. The consumer will use this API key in order to consume this API.
3. The API Key Generation policy publishes the API in the Mediator.
4. If the publication is successful, the API Key Generation policy returns a success message, including data that is pertinent to the published API. (This includes the API key that is required for consuming this API). If the publishing is unsuccessful, the deployer service returns a failure message.
5. The consumer accesses the URL for API consumption, sends the API key as integral part of the HTTP request header or as a query string, and upon validation of the API key consumes the API.
6. If the consumption is successful, the consumer uses the API. If the consumption is unsuccessful for some reasons of authentication, a 500 fault is returned.

How Does Mediator Evaluate Consumers at Run Time?

After you (the API consumer) have successfully registered as a consumer for a particular API, in order to call an API you must pass your API key or OAuth2 access token in your HTTP request header.

- If you use an API key to call the API, the client must pass the API key in the HTTP request header or as a query string parameter. The use of this key establishes the client's identity and authentication.
- If you use an OAuth2 access token to call the API, the client must pass the OAuth2 access token as an integral part of the HTTP request header. An OAuth2 token is a unique token that a client uses to invoke APIs using the OAuth 2.0 protocol. The token contains an identifier that uniquely identifies the client. The use of a token establishes the client's identity, and is used for both the authentication and authorization.

In addition, the API provider can include run-time security actions in the run-time governance rules for APIs. Security actions can validate clients' request and response messages (through WSS X.509 certificates, WSS username tokens, and so on) or identify clients (through IP address or hostname). To enforce client validation, Mediator maintains a list of consumer applications specified in CentraSite that are authorized to access the API published to Mediator. For more information about run-time governance rules, see ["Run-Time Governance Reference" on page 327](#).

How Does a Consumer Use the Generated API Key?

REST services rely on HTTP methods like GET, POST, PUT, and DELETE to make request to an API provider, so the API keys are closely tied to these HTTP methods, where they are sent as part of these HTTP method requests.

CentraSite allows you to set API keys as part of the HTTP header or as the query component of an API request.

Important: In the case where a consumer is sending a request with both credentials (HTTP header) and (query string), the HTTP header take precedence over the query string when the Mediator is determining which credentials it should use for the consumption.

Request Header

The API keys are passed as the HTTP header component of an API consumption request. The HTTP header corresponds to an array of header names to include for that particular API consumption.

The following example demonstrates a typical HTTP request with API keys that form the header value of the API Access URL.

```
x-CentraSite-APIKey:a4b5d569-2450-11e3-b3fc-b5a70ab4288a
```

Query String

The API keys are passed as the query component of an API consumption request.

The following example demonstrates a typical HTTP GET request with API keys that form a query string of the API Access URL.

```
http://localhost:5555/ws/RestAPI?APIKey=a4b5d569-2450-11e3-b3fc-b5a70ab4288a
```

Notice that the API keys are added to the path after a "?", and specified as key-value pair.

What Happens When You Request for API Consumption?

When you request an API for consumption using the Access URL and the generated API key, CentraSite automatically validates the APIs run-time actions to ensure that any "Evaluate" action that appears in the policy governance rule of an API is validated with the consumer requesting for that API.

What Happens When Consumption Fails?

If the API consumption encounters a problem due to one or more of the following reasons, a 500 SOAP fault is returned.

- If the API key value in the HTTP header or the query string is authenticated as invalid.

The sample message looks like this:

```
The request is authenticated as invalid.
```

- If the HTTP header is not present in the request.

The sample message looks like this:

```
A required header is missing in the request.
```

- If the API key value in the HTTP header is expired.

The sample message looks like this:

```
The API key has expired.
```

Fetching and Using Your OAuth2 Access Tokens for Consumption

If you will be using the OAuth 2.0 protocol and you have successfully registered as a consumer for an API, you should have received your OAuth2 client credentials (a `client_id` and `client_secret`).

Now you need to obtain an OAuth2 access token by passing your client credentials to the Mediator-hosted REST service `mediator.oauth2.getOAuth2AccessToken`. This service will provide an OAuth2 access token that you can subsequently include in your requests to call the API.

The service's input parameters are:

- `client_id`
- `client_secret`
- `scope` (optional). The scope value is the name of the virtual service. If the scope value is valid, Mediator obtains the access token. If no scope value is provided, Mediator provides the access token to the scope in which the client is allowed, and adds the scope to the response. To pass the scope, pass it in the request body.

Ways for Clients to Provide the Inputs

There are three ways in which a client can provide the inputs for this service:

- Provide inputs in the Basic authentication header (recommended).

The client can provide the client credentials (`client_id` and `client_secret`) in the Authorization header using the following form:

```
Authorization: Basic <base-64-encoded client_id:password,
client_secret>
```

If you want to pass the scope, pass it in the request body.

- Provide JSON inputs for the service.

The client can send a JSON request to the service in the following form:

```
{
  "client_id" : "",
  "client_secret": "",
  "scope" : ""
}
```

Note: The client should contain the header `Content-type:application/json` in the request.

- Provide inputs in the request body

The OAuth2 specifications do not support sending the client credentials over the URL as URL-Encoded. However, you can send the client credentials in the request body using the following form:

```
client_id=<client_id>&client_secret=<client_secret>&scope=<scope>
```

Note: The client should contain the header `Content-type:application/x-www-form-urlencoded` in the request.

Note: If a client provides the `client_id` and `client_secret` in both the Authorization header and the request body, the credentials given in the Authorization header are used.

Using HTTPS for Granting Access Tokens

For security reasons it is recommended to use HTTPS in your production environment. If you will be using HTTPS as the transport protocol over which the OAuth2 access tokens will be granted authorization, you must set the parameters `pg.oauth2.isHTTPS` and `pg.oauth2.ports` as described in *Administering webMethods Mediator*.

Responses Returned to Clients

Following are sample responses that are returned to the client:

- Sample XML response:

```
<Response
xmlns="https://localhost/rest/pub.mediator.oauth2.getOAuth2AccessToken">
<access_token>db95b40095f31439a1cd8f411e64abe8</access_token>
<expires_in>3600</expires_in>
<token_type>Bearer</token_type>
</Response>
```

- Sample JSON response:

```
{
  "access_token": "db95b40095f31439a1cd8f411e64abe8",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

Managing Your API Keys

An API key is a secret code that you can use to identify yourself to CentraSite when you interact with virtualized API. You generate an API Key in Mediator by registering as consumer for the virtualized API, and then use the key in interactions with the specific

virtualized API published to Mediator. Multiple interactions may be performed with the same API key.

The following sections describe the various operations you can perform on the API key at the disposal of the API provider (owner of the API).

Operations	For Instructions...
View Details of an API Key	See "Fetching Details of Your API Keys" on page 255.
Renew an API Key	See "Renewing Your API Key" on page 257.
Revoke an API Key	See "Revoking Your API Key" on page 258.
Delete an API Key	See "Deleting Your API Key" on page 259.

Fetching Details of Your API Keys

If you will use the API key authentication and you have successfully registered as a consumer for a virtualized API, you should have received your API key details.

The CentraSite Business user interface enables users to view details of an API key in the following ways:

- Through the email notification messages that were auto-generated by CentraSite. For procedures, see ["Viewing Details of an API Key Using the Email Notification"](#) on page 255.
- Through the **Consumer Overview** profile of an API. For procedures, see ["Viewing Details of an API Key Using the API Details Page"](#) on page 256.
- Through the **User Preferences** page for a consumer. For procedures, see ["Viewing Details of an API Key Using the User Preferences"](#) on page 256

Viewing Details of an API Key Using the Email Notification

Once your API registration request is approved, CentraSite sends an automated email message containing details of the API key value and its usage to both the approver and consumer.

If you want to receive email messages of the API key, make sure:

- You have the notification option set as **Email** in the User Preferences page.
- You have specified a valid email address.

Access your mailbox.

The information contained in the email message depends on whether you access the virtualized API as a consumer or the owner.

- If you were the requestor for the virtualized API, you will see the following information:
 - API key
 - Key expiration date
 - API key usage
- If you are the owner of the virtualized API, you will see the following information:
 - API key
 - Key expiration date

Viewing Details of an API Key Using the API Details Page

To view the details of an API key using the native and/or virtualized API details page, proceed as follows:

To view the details of an API key

1. Display the details page for the API whose key value want to view.
2. Select the **Consumer Overview** profile.

The information contained in this profile depends on whether you access the API as a consumer or the owner.

- If you were the requestor for the API, you will see the following information:
 - API key
 - Key expiration date
 - API key usage
- If you are the owner of the API, you will see the following information:
 - API key
 - Key expiration date

Viewing Details of an API Key Using the User Preferences

To view the details of all your API keys using the User Preferences page, proceed as follows:

To view the details of an API key

1. In CentraSite Business UI, click on your username shown in the header at the top of the page.

This opens the User Preferences page.

2. Expand the **My API Keys** section. This displays a list of API keys that are available to you in the CentraSite registry.

Note: The **My API Keys** section IS NOT VISIBLE if you do not have at least one API key.

3. Each entry in the API key list includes:

- API key
- Key value
- Key expiration date

Additionally in the User Preferences page, you can do the following operations on an API key:

- Renew your API key, provided the key has a limited usage period, as described in ["Renewing Your API Key" on page 257](#).
- Revoke your API key temporarily from the CentraSite registry, as described in ["Revoking Your API Key" on page 258](#).
- Delete your API key permanently from the CentraSite registry, as described in ["Deleting Your API Key" on page 259](#).

Renewing Your API Key

API keys have an expiration period, which is set by the API provider. After an API key is generated, sometimes the API consumer might have to renew the old key due to expiration or security concerns. The API provider can also change the expiration period for the API key or set it so that the key never expires. For more information about configuring the API consumption settings for API key authentication, see *Working with the CentraSite Business UI*.

To request for renewal of an API key, the following prerequisites must be met:

- API provider must have configured the predefined policy `API Key Renewal`, which enables the CentraSite Administrator, API Provider or designated approvers to approve or reject the "Renew API Key" requests. For more information about API key renewal policy, see *Working with the CentraSite Business UI*.
- A gateway instance (for example, Mediator) should be up and running. For information on gateways, see ["Run-Time Gateways" on page 59](#).
- The consumer must be a registered consumer for the specified API. For more information on registering as consumers for an API, see ["Consumer Registrations" on page 45](#).

To renew your API key

1. In the **My API Keys** section, displays a list of your API keys in CentraSite.

2. Move the cursor over the API key you want to renew. This causes a **Renew** icon to appear, that you can use for renewing.
3. Click the icon to renew the API key.

Important: If the API key has unlimited expiration period, then this icon will not be displayed for that API key.

Sometimes you might have to require an approval to renew the API key. If your API has the **Require Approval** configured in the API consumption settings, CentraSite will not renew the API key until the required approvals are obtained. However, if an approval workflow is not configured for the API, the API key is renewed immediately. For more information about approval actions, see *Working with the CentraSite Business UI*.

After renewing the API key, CentraSite automatically publishes the API to the Mediator, triggered by a *Deploy Access Key* action that is included in the API Key Renewal policy. For more information about the usage of this policy, see *Working with the CentraSite Business UI*.

Once the API key renewal request is approved by the designated approvers, CentraSite sends an email message to the API consumer informing the new validity of API key.

Revoking Your API Key

The API consumer might want to revoke an API key if, for example, the key is no longer needed or if an error is found in the API.

To request for revocation of an API key, the following prerequisites must be met:

- API provider must have configured the predefined policy `API Key Revocation`, which enables the CentraSite Administrator, API Provider or designated approvers to approve or reject the "Renew API Key" requests.
- A gateway instance (for example, Mediator) should be up and running. For information on gateways, see ["Run-Time Gateways" on page 59](#).

To revoke your API key

1. In the **My API Keys** section, displays a list of your API keys in CentraSite.
2. Move the cursor over the API key you want to revoke. This causes a **Delete** () icon to appear, that you can use for revoking.
3. Click the icon to revoke the API key.

A confirmation message appears that the API key will be revoked.

Once the API key revocation is processed, CentraSite sends an email message to the API Consumer informing the request has been processed successfully.

Deleting Your API Key

The API key consumer can delete API keys. Deleting an API key permanently removes the key from the CentraSite registry/repository. Deleting an API key will not remove the API that is associated with it.

When you delete an API key, CentraSite removes an entry for the API key (that is, it removes the instance of the API key from CentraSite's object database). Also note that:

- An API key can only be deleted if it is already revoked.
- You cannot delete an API key that is in the “pending” mode (for example, awaiting a renew approval).
- You must be a registered consumer for the specified API. For more information on registering as consumers for an API, see ["Consumer Registrations" on page 45](#).

You can delete a single API key or a selected set of API keys. The descriptions in this section give you details on how to do this.

Deleting an Individual API key

The following procedure describes how to delete a single API key.

To delete a single API key

1. In CentraSite Business UI, display the details page for the API key that you want to delete.
2. In the actions bar for the API key, click **Delete**.
3. When you are prompted to confirm the delete operation, click **OK**.

The API key is permanently removed from the CentraSite registry/repository.

Deleting Multiple API Keys in a Single Operation

You can delete multiple API keys in a single step. The rules described above for deleting a single API key apply also when deleting multiple API keys.

To delete a set of API keys in a single step

1. Display the list of API keys.
2. Mark the checkbox next to the name of each API key that you want to delete.
3. In the actions bar of the Search Results page, click **Delete**.
4. When you are prompted to confirm the delete operation, click **OK**.

Each selected API key is permanently removed from the CentraSite registry/repository.

Note: If one or more of the selected APIs is in pending mode (for example, awaiting approval), an error message will appear and no API keys will be deleted.

Privileged User of a Virtualized API

A *Privileged User* is a user who has an elevated level of access to perform various actions on the virtualized API, such as configuring the API's consumption settings, publishing the virtualized API to gateways, and so on.

When the user tries to perform a privileged action on the virtualized API, CentraSite validates the user's ID to determine whether the user holds the necessary privileges, and if so, it validates whether the privileges are enabled. If the user fails these validation, CentraSite does not perform the action.

A user who belongs to the CentraSite Administrator role defines the privileged user for the virtual API in the custom configuration file (centrasite.xml).

```
<Service id="DeploymentService" privilegeUser="INTERNAL\Administrator"
  requiredRoles="Mediator Publisher"> </Service>
```

wherein,

<u>Input Parameter</u>	<u>Specifies...</u>
id	A unique identifier of the virtualized API.
privilegeUser	User who has privilege to access and execute various actions on the virtualized API.
requiredRoles	A set of roles required to access and execute the various actions on the virtualized API.

By default, an `INTERNAL\Administrator` with `Mediator Publisher` role is configured as the privileged user for the virtualized API.

7 Invoking webMethods IS Services in Virtual Services

■ Introduction	262
■ Using the Security API in webMethods IS Services	263

Introduction

A webMethods Integration Server (IS) service is a user-defined Integration Server flow service that you can invoke in:

- Request Processing steps, to preprocess the request message before it is submitted to the native service.
- Response Processing steps, to preprocess the response message from the native service before it is returned to the consuming application.

A webMethods IS service must be running on the same Integration Server as webMethods Mediator. It can call out a C++ or Java or .NET function. It can also call other Integration Server services to manipulate the SOAP message.

The input pipeline for a webMethods IS service should have the following input variables:

- **proxy.name:** This is the name of the virtual service.
- **SOAPEnvelope:** Contains the SOAP envelope. This is of the Java type `org.apache.axiom.soap.SOAPEnvelope`.
- **EnvelopeString:** Contains the SOAP envelope as a string.

Limitation: **EnvelopeString** will *not* be sent to IS services if a request uses the "MTOM" SOAP Optimization Method and if the Integration Server property `watt.server.SOAP.MTOMStreaming.enable` is set to true.

- **MessageContext:** Mediator will automatically place a MessageContext variable into the pipeline before executing the webMethods IS service call. MessageContext is of the Java type `org.apache.axis2.context.MessageContext`.

Integration Server users can use the Axis2 MessageContext object to manipulate the incoming SOAP request. The Integration Server provides built-in services (that is, the `pub.soap.*` services) to work with the MessageContext object to get/set/modify the SOAP body, header, properties, and so on. Integration Server users should use these services to extract the information they need from the MessageContext to build the necessary business logic. Users do not need to understand Axis2 or Axiom (the xml object model based on StAX) to work with the SOAP request, because if they are familiar with the Integration Server `pub.soap` services, they can accomplish most of the tasks. For more information about these related Integration Server services, see the *webMethods Integration Server Built-In Services Reference*.

- **JSONRESTContentString:** Will appear only for REST services with the Content-Type `application/json` or `application/json/badgerfish`. For more information, see ["Multiple Root Nodes in JSON REST Services" on page 320](#).
- **UpdatedJSONRESTContentString:** Will appear only for REST services with the Content-Type `application/json` or `application/json/badgerfish`. For more information, see ["Multiple Root Nodes in JSON REST Services" on page 320](#).

You can use the following constructs in a webMethods IS service:

- Predefined or custom context variables. For more information, see [Using Context Variables in Virtual Services](#).
- The Security API provided by Mediator (for SOAP-based services only). For more information, see ["Using the Security API in webMethods IS Services" on page 263](#).

Using the Security API in webMethods IS Services

Note: This API is for SOAP-based services only.

Mediator provides Java services that you can use to support WS-Security functionality in a webMethods IS service that you invoke in the Request Processing step.

pub.mediator.security.ws:AddUsernameToken

Adds the WS-Username Token 1.0 and 1.1 to the request. This service includes the following input parameters:

Note: For reasons of legibility some of the examples below contain break lines and might not work when pasted into applications or command line tools.

In the parameter descriptions, the data type is listed first, followed by the Java type in parenthesis, for example, "Object (org.apache.axis2.context.MessageContext)".

Input Parameters

<i>username</i>	String (String) (Required) The value that will be added as the Username element in the token. The default value is: ""
<i>MessageContext</i>	Object (org.apache.axis2.context.MessageContext) (Required) Mediator will place a <i>MessageContext</i> variable into the pipeline before executing the webMethods IS service call. The default value is: org.apache.axis2.context.MessageContext instance
<i>password</i>	String (String) (Optional) The password for the token. You must specify <i>password</i> if the <i>passwordType</i> is set to either TEXT or DIGEST. The default value is: ""

<i>passwordType</i>	<p>String (String) (Optional) Specifies how the password will be added in the token. Specify one of the following values:</p> <ul style="list-style-type: none">■ NONE: The password will not be added.■ TEXT: The password is added in plain text.■ DIGEST: The password is added in digested form (as specified in the UsernameToken profile). <p>The default value is: <code>NONE</code></p>
<i>addNonce</i>	<p>Boolean (Boolean) (Optional) Specifies whether the Nonce element will be added to the token.</p> <p>The default value is: <code>false</code></p>
<i>addCreated</i>	<p>Boolean (Boolean) (Optional) Specifies whether the Created element will be added to the token</p> <p>The default value is: <code>false</code></p>
<i>salt</i>	<p>byte[] (byte[]) (Optional) The value for the /wsse11:UsernameToken/wsse:Salt element. Its value is a 128 bit number serialized as xs:base64Binary.</p> <p>The default value is: <code>null</code></p>
<i>iteration</i>	<p>int (Integer) (Optional) Indicates the number of times the hashing operation is repeated when deriving the key. It is expressed as a xs:unsignedInteger value. If it is not present, a value of 1000 is used for the iteration count.</p> <p>The default value is: <code>1000</code></p>
<i>useMac</i>	<p>Boolean (Boolean) (Optional) Indicates if the derived key will be used as a Message Authentication Code (MAC) or as a symmetric key for encryption.</p> <p>The default value is: <code>false</code></p>
<i>useBasicAuthCredentials</i>	<p>Boolean (Boolean) (Optional) If this parameter is set to <code>true</code>, Mediator will try to use the username and password from the "Authorization" HTTP header. In this case the 'username' and 'password' fields need not be specified.</p> <p>The default value is: <code>false</code></p>
<i>actor</i>	<p>String (String) (Optional) Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP</p>

request. If the request already has a security header with the actor specified in it, then this value will not overwrite it.

The default value is: ""

mustUnderstand **Boolean (Boolean)** (Optional) Specifies whether the security header will have the `mustUnderstand` attribute set to 0 or 1 (false / true). If the security header already has this attribute set, this value will not overwrite it.

The default value is: `false`

pub.mediator.security.ws:AddX509Token

Adds a X.509 certificate (or certificate chain) as a BinarySecurityToken (BST) element in the outbound SOAP request. This service includes the following input parameters:

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

In the parameter descriptions, the data type is listed first, followed by the Java type in parenthesis, for example, "Object (org.apache.axis2.context.MessageContext)".

Input Parameters

MessageContext **Object (org.apache.axis2.context.MessageContext)** (Required)
Mediator will place a *MessageContext* variable into the pipeline before executing the webMethods IS service call.

The default value is: org.apache.axis2.context.MessageContext instance

keystoreFile **String (String)** (Required) The absolute path to a keystore file on the system where Mediator is running.

The default value is: ""

keystorePassword **String (String)** (Required) The password for the keystore.

The default value is: ""

keystoreType **String (String)** (Optional) The type of keystore represented by the file (can be `JKS`, `JCEKS`, or `PKCS12`).

The default value is: `JKS`

<i>keyAlias</i>	String (String) (Required) The key alias whose X509 certificate will be sent in the soap request as a BST. The default value is: ""
<i>useCertificatePath</i>	Boolean (Boolean) (Optional) If set to <code>true</code> will use the entire certificate chain represented by the key alias instead of just a single certificate. The default value is: <code>false</code>
<i>actor</i>	String (String) (Optional) Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it. The default value is: ""
<i>mustUnderstand</i>	Boolean (Boolean) (Optional) Specifies whether the security header will have the <code>mustUnderstand</code> attribute set to 0 or 1 (false / true). If the security header already has this attribute set, then this value will not overwrite it. The default value is: <code>false</code>

pub.mediator.security.ws:AddSamlSenderVouchesToken

This service enables a Security Token Service (STS) client to send a WS-Trust request to a configured STS to obtain a SAML v1/v2 assertion. For the details about configuring Mediator to act as an STS client, see *Administering webMethods Mediator*.

This service adds the obtained SAML assertion to the original request that is sent by the client to the native service, and includes the following parameters.

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

In the parameter descriptions, the data type is listed first, followed by the Java type in parenthesis, for example, "Object (org.apache.axis2.context.MessageContext)".

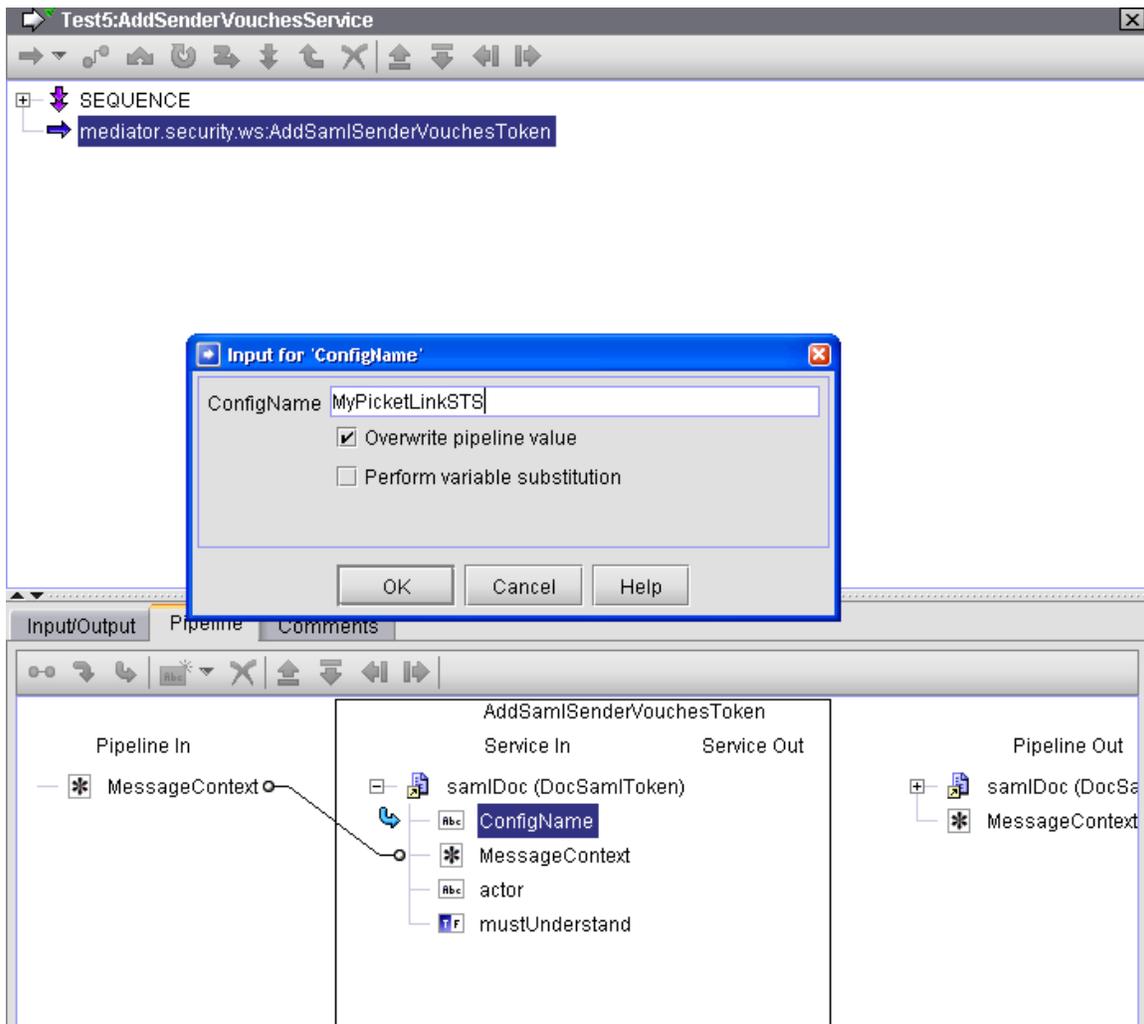
Input Parameters

<i>ConfigName</i>	String (String) (Required) References a previously configured STS configuration name. The default value is: ""
-------------------	--

<i>MessageContext</i>	Object (org.apache.axis2.context.MessageContext) (Required) Mediator will place a <i>MessageContext</i> variable into the pipeline before executing the webMethods IS service call. The default value is: org.apache.axis2.context.MessageContext instance
<i>addTimeStamp</i>	Boolean (Boolean) (Optional) Adds a Timestamp element (with the duration specified in <i>timeToLive</i>) to the WS-Security header of the request, and includes it in the signature. (The other items that are signed are the body and SAML assertion.) The default value is: false
<i>timeToLive</i>	Integer (Integer) (Optional) If <i>addTimeStamp</i> is true, <i>timeToLive</i> specifies the duration (in seconds) for which the request is valid. The default value is: 300 (5 minutes)
<i>actor</i>	String (String) (Optional) Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it. The default value is: ""
<i>mustUnderstand</i>	Boolean (Boolean) (Optional) Specifies whether the security header will have the mustUnderstand attribute set to 0 or 1 (false / true). If the security header already has this attribute set, then this value will not overwrite it. The default value is: false

Example of using AddSamlSenderVouchesToken

The sample service shown below is configured by providing the *MessageContext* and *ConfigName* parameters. The value of *ConfigName* must be the name of a previously configured STS name, which is configured on the Mediator Configuration page.



pub.mediator.security.ws:AddTimestamp

Adds a timestamp to the outbound SOAP request WS-Security header. This service includes the following input parameters:

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

In the parameter descriptions, the data type is listed first, followed by the Java type in parenthesis, for example, "Object (org.apache.axis2.context.MessageContext)".

Input Parameters

timeToLive **Integer (Integer)** (Optional) Specifies the duration (in seconds) for which the request is valid.

The default value is: 300 (5 minutes)

signTimestamp **Boolean (Boolean)** (Optional) Indicates whether the generated timestamp must be signed by Mediator using the configured keystore and signing alias.

Note: For *signTimestamp* to work, you must ensure that a valid IS keystore and signing alias are configured in Mediator. For details, see *Administering webMethods Mediator*.

The default value is: false

useMilliSecond Precision **Boolean (Boolean)** (Optional) Indicates whether the generated timestamp must have millisecond precision.

The default value is: true

MessageContext **Object (org.apache.axis2.context.MessageContext)** (Required) Mediator will place a *MessageContext* variable into the pipeline before executing the webMethods IS service call.

The default value is: org.apache.axis2.context.MessageContext instance

actor **String (String)** (Optional) Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it.

The default value is: ""

mustUnderstand **Boolean (Boolean)** (Optional) Specifies whether the security header will have the mustUnderstand attribute set to 0 or 1 (false / true). If the security header already has this attribute set, then this value will not overwrite it.

The default value is: false

pub.mediator.addressing:AddWSAddressingHeaders

Adds WS-Addressing headers to a SOAP request sent by the client before Mediator forwards the request to the native service.

This service includes the following input parameters:

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

In the parameter descriptions, the data type is listed first, followed by the Java type in parenthesis, for example, "Object (org.apache.axis2.context.MessageContext)".

Input Parameters

isVersionSubmission

Boolean (Boolean) (Optional) The WS-Addressing version that should be used.

- If true, the WS-Addressing submission namespace will be used.

```
http://schemas.xmlsoap.org/ws/2004/08/addressing
```

- If false, the Final specification namespace will be used.

```
http://www.w3.org/2005/08/addressing
```

The default value is: `false`

To

String (String) (Optional) This value corresponds to the `/wsa:To` addressing header. You must specify a value that corresponds to the destination of the request message.

If this value is not specified, the default value depends on the *isVersionSubmission* property value. One of the following anonymous EPR values will be sent:

- If *isVersionSubmission* is set to `true`, the anonymous EPR value is:

```
http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
```

- If *isVersionSubmission* is set to `false`, the anonymous EPR is:

```
http://www.w3.org/2005/08/addressing/anonymous
```

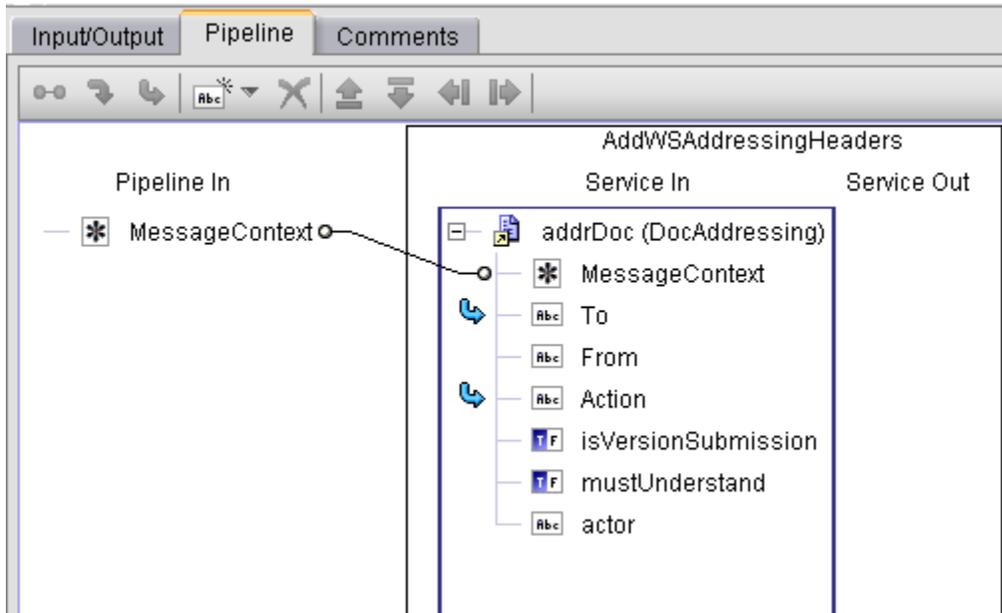
From

String (String) (Optional) This value corresponds to the `/wsa:From` addressing header and refers to the source of the message.

	The default value is: ""
<i>Action</i>	<p>String (String) (Optional) This value corresponds to the / <code>wsa:Action</code> addressing header. By default, this property has the same value as the operation on the virtual service being invoked (which will usually correspond to the same operation on the native service). But the user can specify a different value corresponding to the native service being called.</p> <p>The default value is: URI identifying input operation corresponding to a WSDL port type being called on the virtual service</p>
<i>MessageContext</i>	<p>Object (org.apache.axis2.context.MessageContext) (Required) Mediator will place a <i>MessageContext</i> variable into the pipeline before executing the webMethods IS service call.</p> <p>The default value is: org.apache.axis2.context.MessageContext instance</p>
<i>actor</i>	<p>String (String) (Optional) Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it.</p> <p>The default value is: ""</p>
<i>mustUnderstand</i>	<p>Boolean (Boolean) (Optional) Specifies whether the security header will have the <code>mustUnderstand</code> attribute set to 0 or 1 (false / true). If the security header already has this attribute set, this value will not overwrite it.</p> <p>The default value is: false</p>

Example of using AddWSAddressingHeaders

The sample service shown below is configured by providing the `MessageContext` parameter.



8 Using Context Variables in Virtual Services

■ Introduction to Context Variables in Virtual Services	274
■ The Predefined Context Variables	274
■ The API for Context Variables	279

Introduction to Context Variables in Virtual Services

Mediator provides predefined context variables, and you can declare your own custom context variables. You can use both predefined and custom context variables when you configure various processing steps of a virtual service. Specifically, you can use them:

- In a webMethods IS service that you create and invoke during the **Request Processing** step or the **Response Processing** step.
- In a routing rule that you create in the **Context-Based Routing Protocols** step.

The Predefined Context Variables

You can use the predefined context variables listed below. Any context variable state defined during the inbound request processing steps will still be available during the outbound response processing steps.

- Note:** Keep the following points in mind:
- To set, get or remove the predefined context variables, use [The API for Context Variables](#) provided by Mediator.
 - You do not need to declare the predefined context variables. If you attempt to declare an existing predefined context variable, an error will occur.

Context Variable Display Name and Variable Name	Description
Average Response AVG_SUCCESS_TIME	The average amount of time it took the service to complete all invocations in the current interval. This is measured from the moment Mediator receives the request until the moment it returns the response to the caller.
	<p>Note:By default, Average Response Time does not include metrics for failed invocations. You can include metrics for failed invocations by setting the <code>pg.PgMetricsFormatter.includeFaults</code> parameter to true. For details about advanced settings, see <i>Administering webMethods Mediator</i>.</p>
Client IP Address	The IP address used to send the request to Mediator.

Context Variable Display Name and Variable Name	Description
INBOUND_IP	
Consumer CONSUMER_APPLICATION	The name of the consumer application accessing the service, if known.
Fault Count INTERVAL_FAULT_COUNT	The number of service faults for the interval.
Inbound Content Type MESSAGE_TYPE	A Content-Type defined in axis2.xml for a message formatter. This value must be a key in the axis2 message formatters list, since it is used to control message serialization. (The valid choices are defined as attributes of <messageFormatters/> group in the Integration Server's axis2.xml configuration.)
Inbound HTTP Method INBOUND_HTTP_METHOD	The HTTP method used by the client to send the request (GET, POST, PUT, DELETE, Use Context Variable).
Inbound Protocol INBOUND_PROTOCOL	The protocol (HTTP or HTTPS) of the request.
Maximum Response SLOWEST_SUCCESS_INVOKE	Maximum Response Time.
	<p>Note:By default, Maximum Response Time does not include metrics for failed invocations. You can include metrics for failed invocations by setting the <code>pg.PgMetricsFormatter.includeFaults</code> parameter to true. For details about advanced settings, see <i>Administering webMethods Mediator</i>.</p>
Mediator Host Name MEDIATOR_HOSTNAME	Mediator host name.
Mediator IP Address MEDIATOR_IP	Mediator IP address.

Context Variable Display Name and Variable Name	Description
Mediator Target Name TARGET_NAME	Mediator target name.
Minimum Response FASTEST_SUCCESS_INVOKE	Minimum Response Time. Note: By default, Minimum Response Time does not include metrics for failed invocations. You can include metrics for failed invocations by setting the <code>pg.PgMetricsFormatter.includeFaults</code> parameter to true. For details about advanced settings, see <i>Administering webMethods Mediator</i> .
Outbound HTTP Method ROUTING_METHOD	The HTTP method to be sent to the native service if the inbound HTTP method is custom. Otherwise, this value will be null. For more information, see " Changing the HTTP Method of a REST or XML Request " on page 305.
Success Count INTERVAL_SUCCESS_COUNT	The number of success counts for a given service.
Total Count INTERVAL_TOTAL_COUNT	The total number of counts for a given service.
Virtual Service Name SERVICE_NAME	Virtual service name.
N/A BUILDER_TYPE	A Content Type defined in axis2.xml for a message builder. This value must be a key in the axis2 message builders list, since it is used to control building of native service response messages. (The valid choices are defined as attributes of <messageBuilders/> group in the Integration Server's axis2.xml configuration.)
N/A, no display name INBOUND_REQUEST_URI	A partial reference to a virtual service (for HTTP/HTTPS only). The protocol, host and

Context Variable Display Name and Variable Name	Description
	<p>port are not part of the value. For example, if the following virtual service is invoked:</p> <pre>http://mcusawco:5555/ws/TC1</pre> <p>then the expected value of this variable would be <code>/ws/TC1</code>.</p> <p>For a REST or XML service, the URL might also include query string parameters. For example, if the following virtual service is invoked:</p> <pre>http://mcusawco:5555/ws/cars?vin=1234</pre> <p>the expected value of this variable would be <code>/ws/cars?vin1234</code>.</p> <p>This is useful to know because by the time you are able to access the request inside of Mediator, the REST request would contain a top-level element that looks like this:</p> <pre><vin>1234</vin></pre> <p>So it is not obvious from an XSLT expression or a webMethods IS service callout what part of a REST request came in as a query parameter.</p> <p>Therefore, using this variable along with <code>INBOUND_HTTP_METHOD</code> and <code>INBOUND_PROTOCOL</code>, you can determine the exact entry point URI that was used when a virtual service was invoked.</p>
N/A, no display name	<p>The reason returned by the native service provider in the case where it produced a SOAP fault. This will not contain Mediator errors such as security policy enforcement errors. This variable will only contain the “reason” text wrapped in a SOAP fault.</p> <div data-bbox="727 1602 1370 1913" style="background-color: #f0f0f0; padding: 10px;"> <p>Note:When you are using this variable in Conditional Error Processing message that you are specifying in the Response Processing Step, note the following. If a request is denied due to security policy enforcement, the fault handler variable <code>\$ERROR_MESSAGE</code> variable would contain a native service provider error message or other error messages that</p> </div>

Context Variable Display Name and Variable Name	Description
	result from enforced security assertions. However, \$NATIVE_PROVIDER_ERROR will be null in this case.
N/A, no display name OPERATION	The virtual service operation selected to execute a request.
N/A, no display name PROTOCOL_HEADERS	Contains a map of key-value pairs in the request, where the values are typed as strings. To get/set this variable, see <code>pub.mediator.ctxvar:getContextVariable</code> .
N/A, no display name SOAP_HEADERS	(For use in webMethods IS services only.) Contains an array of the SOAP header elements in the request. To get/set this variable, see <code>pub.mediator.ctxvar:getContextVariable</code> .
N/A, no display name USER	The value defined for the Integration Server session executing the request message. If the request is not authenticated, it will use a default unprivileged account. Otherwise, it will set the Integration Server session to the user credentials used for transport security. Also, if credentials were included for message based security (for example, an X509 token was included and this certificate was mapped to an Integration Server user), then this information would override any transport security (for example, basic authentication).
Note:	<p>When you use predefined context variables in a Conditional Error Processing message in The Response Processing Step (REST/XML), note the following:</p> <ul style="list-style-type: none"> ■ To reference a predefined context variable in a Conditional Error Processing message, you need to prepend a \$ symbol to the context variable name to indicate that variable's value should be referenced. Think of this usage as being similar to the '&' address operation for C variables. A predefined context variable expression might look like this: <code>\$USER="Administrator:"</code> ■ The \$ reference symbol may appear in the text as needed. (for example, as a currency symbol). There is no escape concept used with this operator. That is, no special meaning is attached to two occurrences of this symbol: "\$\$".

- If no value is defined for a valid context variable reference, the string is left unmodified for that context variable.

The API for Context Variables

Mediator provides an API that you can use to:

- Set, get, declare and remove custom context variables.
- Set and get the predefined context variables. (It is not necessary (or even legal) to declare or remove the predefined context variables.)

Mediator provides the following Java services, which are defined in the class `ISMediatorRuntimeFacade.java`.

- `pub.mediator.ctxvar:getContextVariable`
- `pub.mediator.ctxvar:setContextVariable`
- `pub.mediator.ctxvar:declareContextVariable`
- `pub.mediator.ctxvar:removeContextVariable`

Sample flow services are described in this section as well.

- [Sample Flow Service: Getting a Context Variable Value](#)
- [Sample Flow Service: Setting a Context Variable Value](#)

`pub.mediator.ctxvar:getContextVariable`

Use this Java service to retrieve a context variable's value and assign it to a pipeline variable. All parameter names are case-sensitive.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by Mediator.	N/A
varName	in	String	Context variable name (predefined or custom).	PROTOCOL_HEADERS SOAP_HEADERS mx: CUSTOM_VAR

Parameter	Pipeline Type	Data Type	Description	Examples
serValue	out	Object ref	Java.io.Serializable value. (Usually a string).	

Notes on Getting and Setting the `PROTOCOL_HEADERS` and `SOAP_HEADERS` Variables

All context variable values are typed as either "string" or "int" except for the predefined context variables `PROTOCOL_HEADERS` and `SOAP_HEADERS`, which are of the type "IData". You can set/get values for `PROTOCOL_HEADERS` and `SOAP_HEADERS` in one of two ways:

■ Set/get the entire structure.

To set the entire structure, you must:

- Set the `varName` parameter in `pub.mediator.ctxvar:setContextVariable` to `PROTOCOL_HEADERS` or `SOAP_HEADERS`.
- Use the method `ISMediatorRuntimeFacade.setContextVariableValue()`.

To get the entire structure, you must:

- Set the `varName` parameter in `pub.mediator.ctxvar:getContextVariable` to `PROTOCOL_HEADERS` or `SOAP_HEADERS`.
- Use the method `ISMediatorRuntimeFacade.getContextVariableValue()`.

If `varName` is set to `PROTOCOL_HEADERS`, you will get/set the entire `IData` structure containing all of the transport headers. The key is the transport header name (for example, `Content-Type`) and the value is a `String`. The `IData` object for `PROTOCOL_HEADERS` will contain a set of string values where each `IData` string key matches the header name in the transport headers map. The set of possible keys includes the HTTP v1.1 set of headers as well as any custom key-value pairs you might have defined.

If `varName` is set to `SOAP_HEADERS`, you will get/set the entire `IData` structure containing all of the SOAP headers in the SOAP envelope. The key is the array position starting with '0', and the value is an `Axiom OMElement` containing that SOAP header block.

Alternatively, you can set the `varName` parameter to address a specific element in the array. For example, setting it to `PROTOCOL_HEADERS[Content-Type]` would apply to the `Content-Type` transport header. Similarly, setting it to `SOAP_HEADERS[0]` would return a `String` representation of the first SOAP header block (as opposed to an `Axiom OMElement`).

■ Set/get a nested value.

Set a nested value in one of the following ways:

- Set the `varName` parameter in `pub.mediator.ctxvar:setContextVariable` to `PROTOCOL_HEADERS[arrayElement]`, where `[arrayElement]` refers to a specific element. For example, `PROTOCOL_HEADERS[Content-Type]` or `SOAP_HEADERS[0]` (to indicate the first array element in the set).
- Alternatively, use the method `ISMediatorRuntimeFacade.setContextVariableValue()`. You would use this method only if you are writing a Java service and you want to access it through the Java source code.

Get a nested value in one of the following ways:

- Set the `varName` parameter in `pub.mediator.ctxvar:getContextVariable` to `PROTOCOL_HEADERS[arrayElement]`, where `[arrayElement]` refers to a specific element. For example, `PROTOCOL_HEADERS[Content-Type]` or `SOAP_HEADERS[0]` (to indicate the first array element in the set).
- Alternatively, use the method `ISMediatorRuntimeFacade.getContextVariableValue()`. You would use this method only if you are writing a Java service and you want to access it through the Java source code.

You can set/get a nested value inside `PROTOCOL_HEADERS` and `SOAP_HEADERS` via an additional `keyName`. In this case, the object reference will *not* be an `IData` object.

- For `PROTOCOL_HEADERS`, the `keyName` must match the transport header name in a case-sensitive manner (for example, `PROTOCOL_HEADERS[Content-Type]` or `PROTOCOL_HEADERS[Authorization]`). In this case, the `Serializable` value will be a string.
- For `SOAP_HEADERS`, the `keyName` must match the 0-based array element. If a request has a SOAP security header element (that is, `</wsse:Security>`), then it would be addressed as `SOAP_HEADERS[0]`. In this case, the element will be in its string format.

pub.mediator.ctxvar:setContextVariable

Use this Java service to set a value on a context variable. The pipeline variable containing the context variable value should be an object reference that implements `java.io.Serializable`. All parameter names are case-sensitive.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by Mediator.	N/A

Parameter	Pipeline Type	Data Type	Description	Examples
varName	in	String	Context variable name (predefined or custom).	PROTOCOL_HEADERS SOAP_HEADERS mx:CUSTOM_VAR
serValue	in	Object ref	Java.io.serializable value. (Usually a string).	

pub.mediator.ctxvar:declareContextVariable

Use this Java service to declare a *custom* context variable. All custom-defined context variables must be declared in a custom namespace that is identified by using the prefix `mx` (for example, `mx:CUSTOM_VARIABLE`). All parameter names are case-sensitive.

Note: It is not legal to use this service to declare the predefined context variables; you can only declare custom variables.

Parameter	Pipeline Type	Data Type	Description
ctxVar	in	Object ref	The document type defining the context variable object. Use the <code>ctxVar</code> Document Type provided in the Java service <code>pub.mediator.ctxvar:ctxVar</code> and map it to this input variable. Define the name (for example, <code>mx:CUSTOM_VARIABLE</code>), the <code>schema_type</code> (string or int), and <code>isReadOnly</code> (true or false).
ctxVar	out	Object ref	The set Context variable document type.
varNameQ	out	Object ref	<code>javax.xml.namespace.QName</code> value. The <code>QName</code> of the variable.

Note the following:

- After declaring the context variable, you can use the `setContext` variable to set a value on the context variable.
- You do *not* need to declare the following kinds of context variables:
 - The predefined context variables provided by Mediator. If you attempt to declare an existing predefined context variable, an error will occur.
 - Any custom context variable that you define in a routing rule that you create in the context-based routing step.
- Any custom context variables that you explicitly declare in source code using the API will have a declaration scope of `SESSION`.
- Any custom context variable's state that is defined during the inbound request processing steps will still be available during the outbound response processing steps.
- All context variable values are typed as either "string" or "int" (excluding the `SOAP_HEADERS` and `PROTOCOL_HEADERS` variables, which are of the type "IData").
- Valid names should be upper case (by convention) and must be a valid Java Identifier. In general, use alpha-numeric, `$` or `_` symbols to construct these context names. Names with punctuation, whitespace or other characters will be considered invalid and will fail deployment.
- All custom context variables must be declared in a custom namespace that is identified by using an `mx` prefix (for example, `mx:CUSTOM_VARIABLE`).
- To reference a custom context variable in a flat string, you need to prepend a `$` symbol to the context variable name to indicate that variable's value should be referenced. Think of this usage as being similar to the `'&'` address operation for C variables.

An expression that references a custom context variable might look like this:

```
$mx:TAXID=1234 or $mx:ORDER_SYSTEM_NAME="Pluto"
```

Notice that the values of the data type "int" are not enclosed in quotation marks, while the values of the data type "string" are. The quotation marks are only needed if a context variable *expression* (as opposed to a reference) is defined.

- Referencing an undefined context variable does not result in an error.
- Once a variable has been declared it cannot be declared again.

pub.mediator.ctxvar:removeContextVariable

Use this Java service to remove a *custom* context variable from a request/response session. All parameter names are case-sensitive.

Note: Keep the following points in mind:

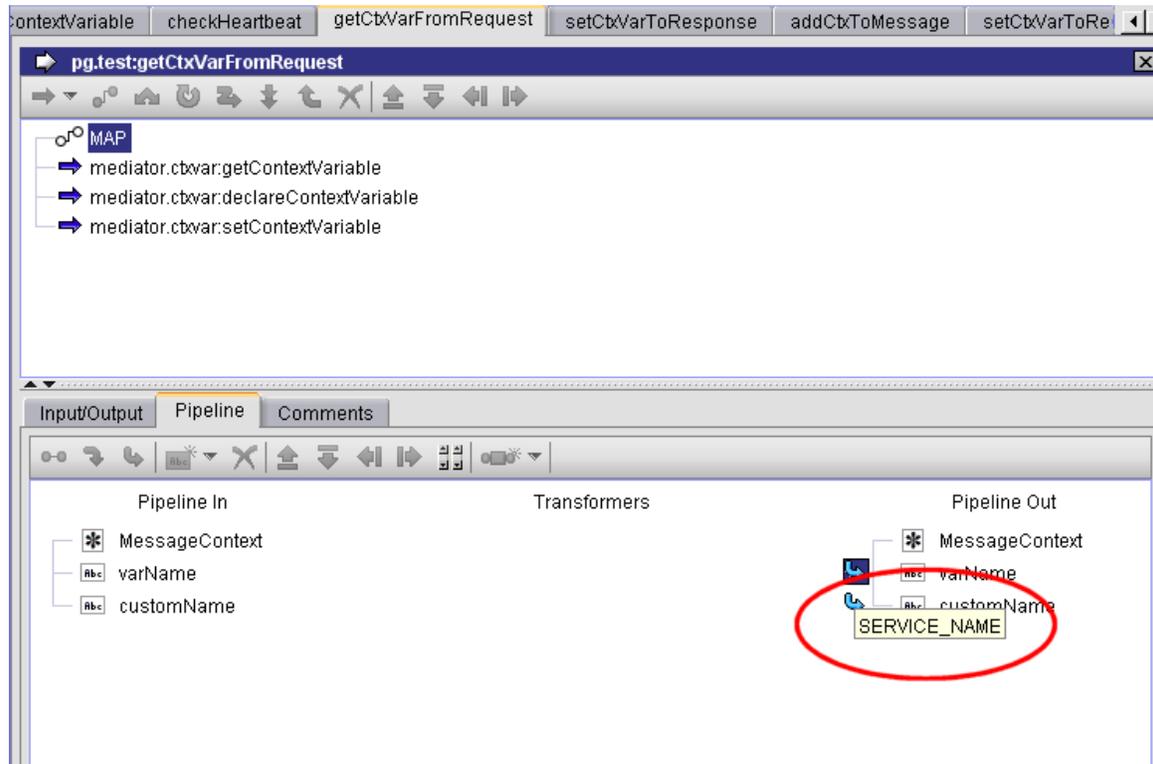
- It is not legal to use this service to remove any predefined context variables; you can only remove custom variables.
- Attempting to remove a non-existent context variable will *not* result in an error.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by Mediator.	N/A
varName	in	String	Custom context variable name.	<code>mx: CUSTOM_VAR</code>

Sample Flow Service: Getting a Context Variable Value

This flow service gets the value of a custom context variable from a request. The service declares a context variable using the name defined in the pipeline variable `customName` (that is, `mx: COMP_TEST`). It is hard-coded to store the predefined context variable `SERVICE_NAME` in this custom context variable name.

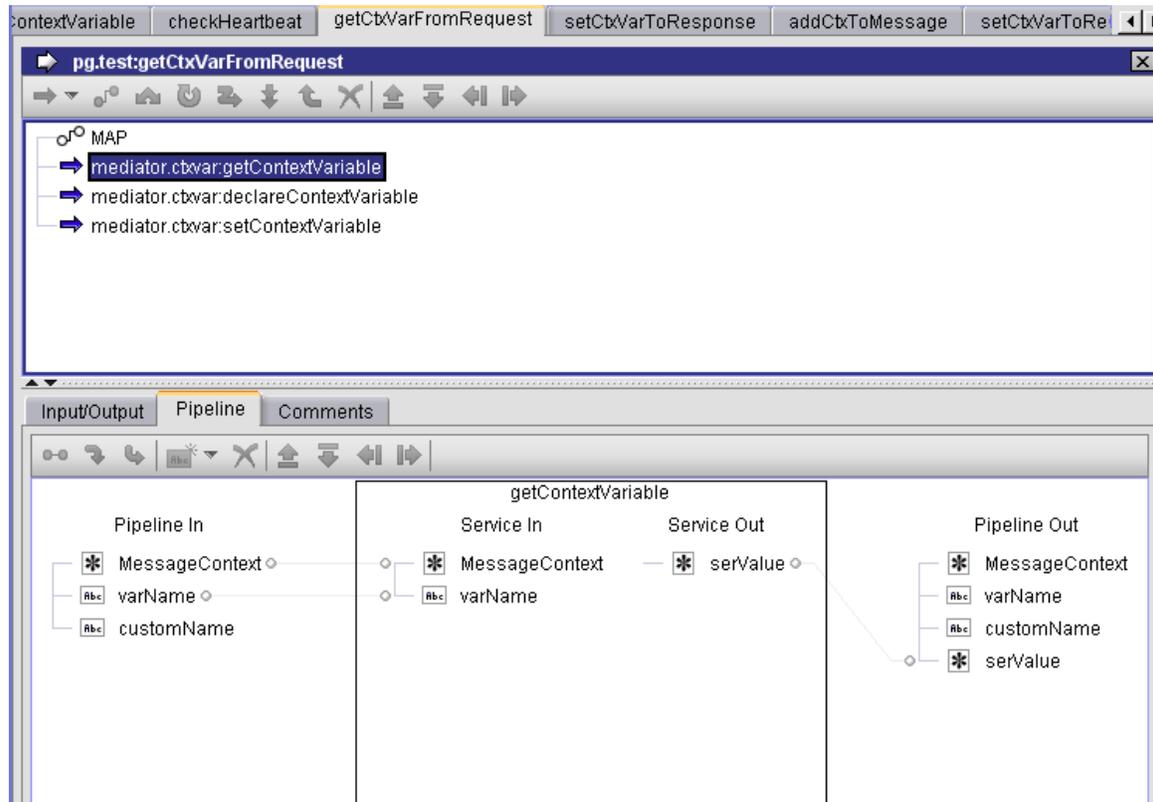
Step 1. Setting varName and customName



In this flow service, the following pipeline variables that are hard-coded as follows:

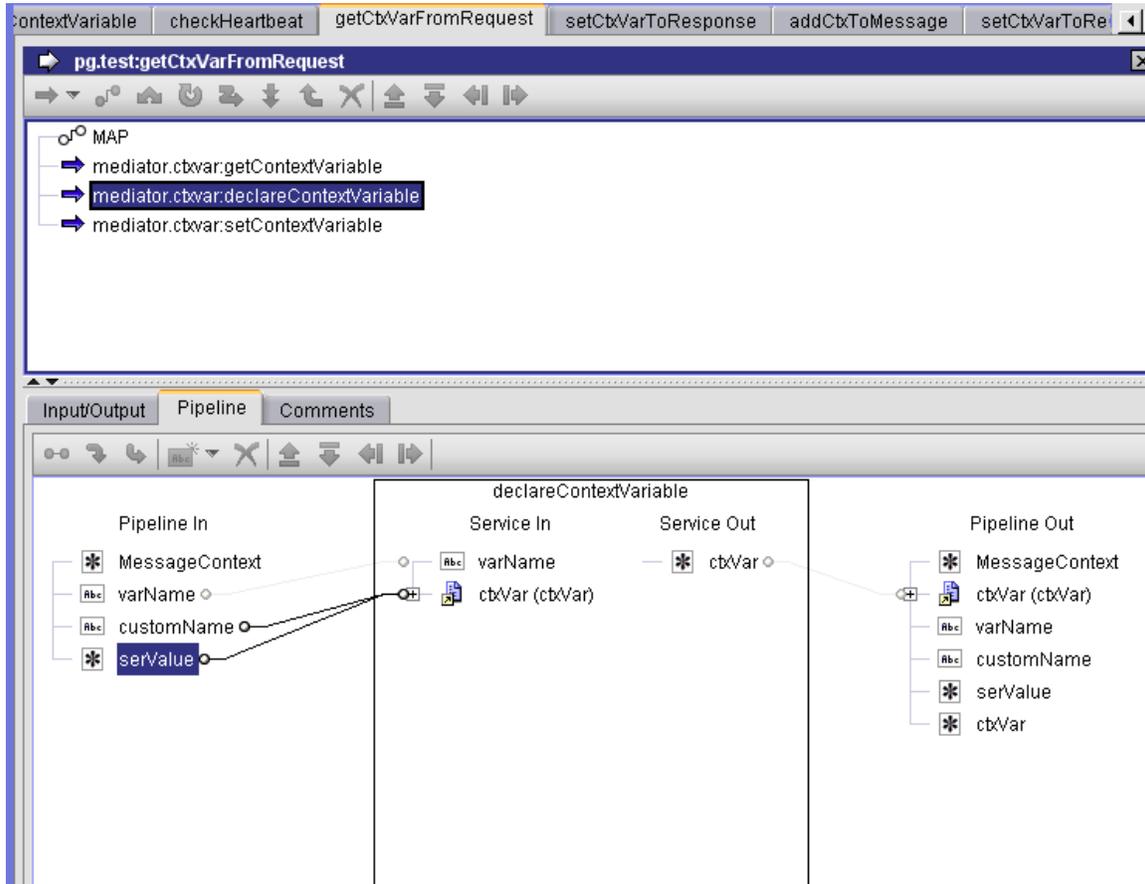
- varName is set to the predefined context variable SERVICE_NAME.
- customName is set to mx:COMP_TEST. SERVICE_NAME is stored in customName in the "Pipeline Out".

Step 2. Calling the `pub.mediator.ctxvar:getContextVariable` service



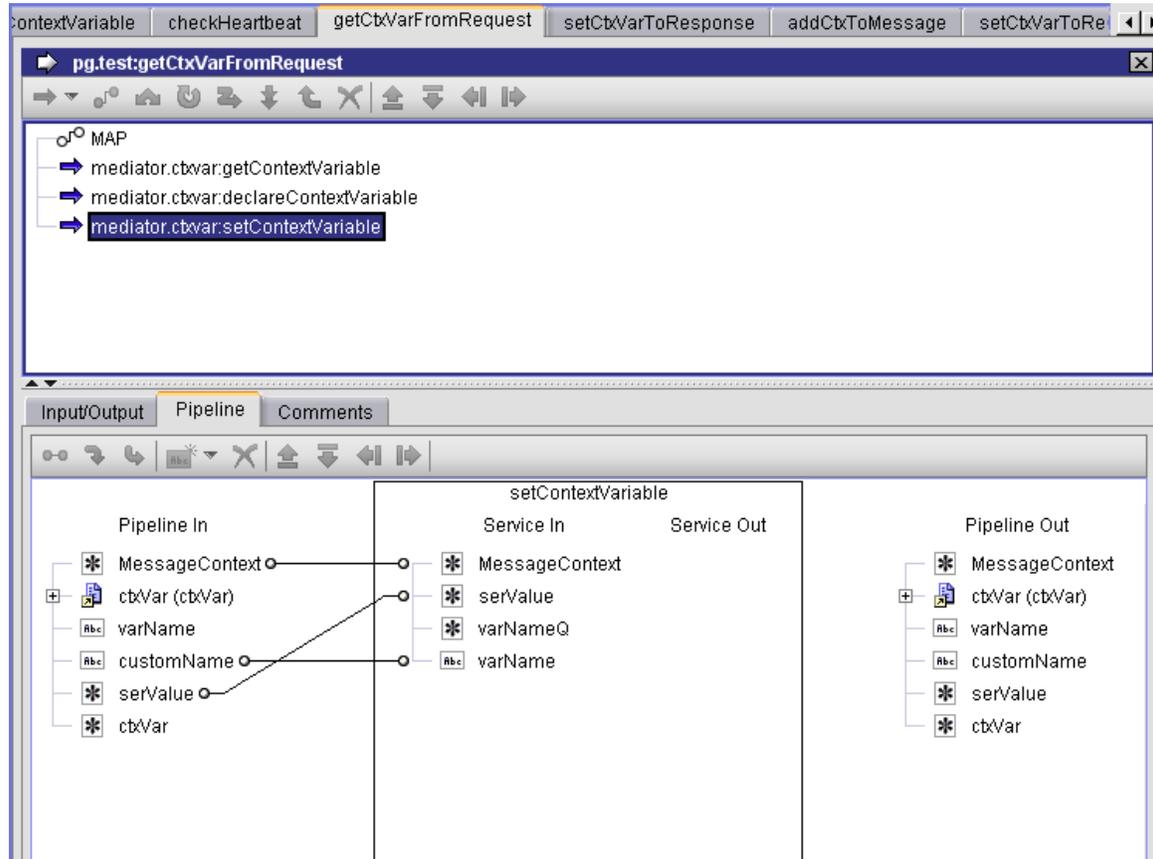
The call to `pub.mediator.ctxvar:getContextVariable` looks up the context variable value for the "Pipeline In" variable `varName` and sets the `Serializable` value in the output pipeline variable named `serValue`.

Step 3. Calling the `pub.mediator.ctxvar:declareContextVariable` service



The call to `pub.mediator.ctxvar:declareContextVariable` takes the serialized value we looked up for `varName` and assigns it to a newly declared custom context variable named `ctxVar`.

Step 4. Calling the `pub.mediator.ctxvar:setContextVariable` service



The call to `pub.mediator.ctxvar:setContextVariable` sets the context variable value back into `MessageContext` so it will be available for the rest of the service invocation steps.

Sample Flow Service: Setting a Context Variable Value

This flow service sets the value of a custom context variable to be used in a response.

This flow service declares a pipeline variable named `customName`, which is set to the value `mx:COMP_TEST`.

This flow service will retrieve the context variable for `customName` and create an element for its context variable value in the response message return to the consumer.

Step 1. Declaring customName

The screenshot displays the configuration for a pipeline step named `pg.test:setCtxVarToResponse`. The top toolbar includes buttons for `contextVariable`, `checkHeartbeat`, `getCtxVarFromRequest`, `setCtxVarToResponse`, and `addCtxToMessage`. The main configuration area shows a **SEQUENCE** of steps:

- MAP
- mediator.ctxvar:getContextVariable
- pg.test:addCtxToMessage

Below the step list, the **Input/Output** tab is active. It features a **Specification Reference** field, **Input** and **Output** dropdown menus, and checkboxes for **Validate input** and **Validate output**. The **Input** section lists the following variables:

- proxy.name
- SOAPEnvelope
- MessageContext
- EnvelopeString
- varName
- customName
- UpdatedSoapRequest

The **Output** section lists the following variables:

- UpdatedSoapRequest
- EnvelopeString
- customName
- UpdatedSoapRequest

We define the `customName` variable value to be `mx:COMP_TEST` so we can use this variable to lookup the custom variable name that was seeded in the previous example.

Step 2. Setting customName to mx:COMP_TEST

The screenshot displays the Oracle Service Bus console interface. At the top, a tabbed window titled 'pg.test:setCtxVarToResponse' is active, showing a sequence of steps: SEQUENCE, MAP, mediator.ctxvar.getContextVariable, and pg.test.addCtxToMessage. Below this, the 'Pipeline' tab is selected, showing a comparison between 'Pipeline In' and 'Pipeline Out' variables. The 'Pipeline In' variables include proxy.name, SOAPEnvelope, MessageContext, EnvelopeString, varName, customName, and UpdatedSoapRequest. The 'Pipeline Out' variables include proxy.name, SOAPEnvelope, MessageContext, EnvelopeString, varName, customName, and UpdatedSoapRequest. The 'customName' variable in the 'Pipeline Out' section is highlighted with a blue selection box, and a blue arrow points to it from the 'customName' variable in the 'Pipeline In' section.

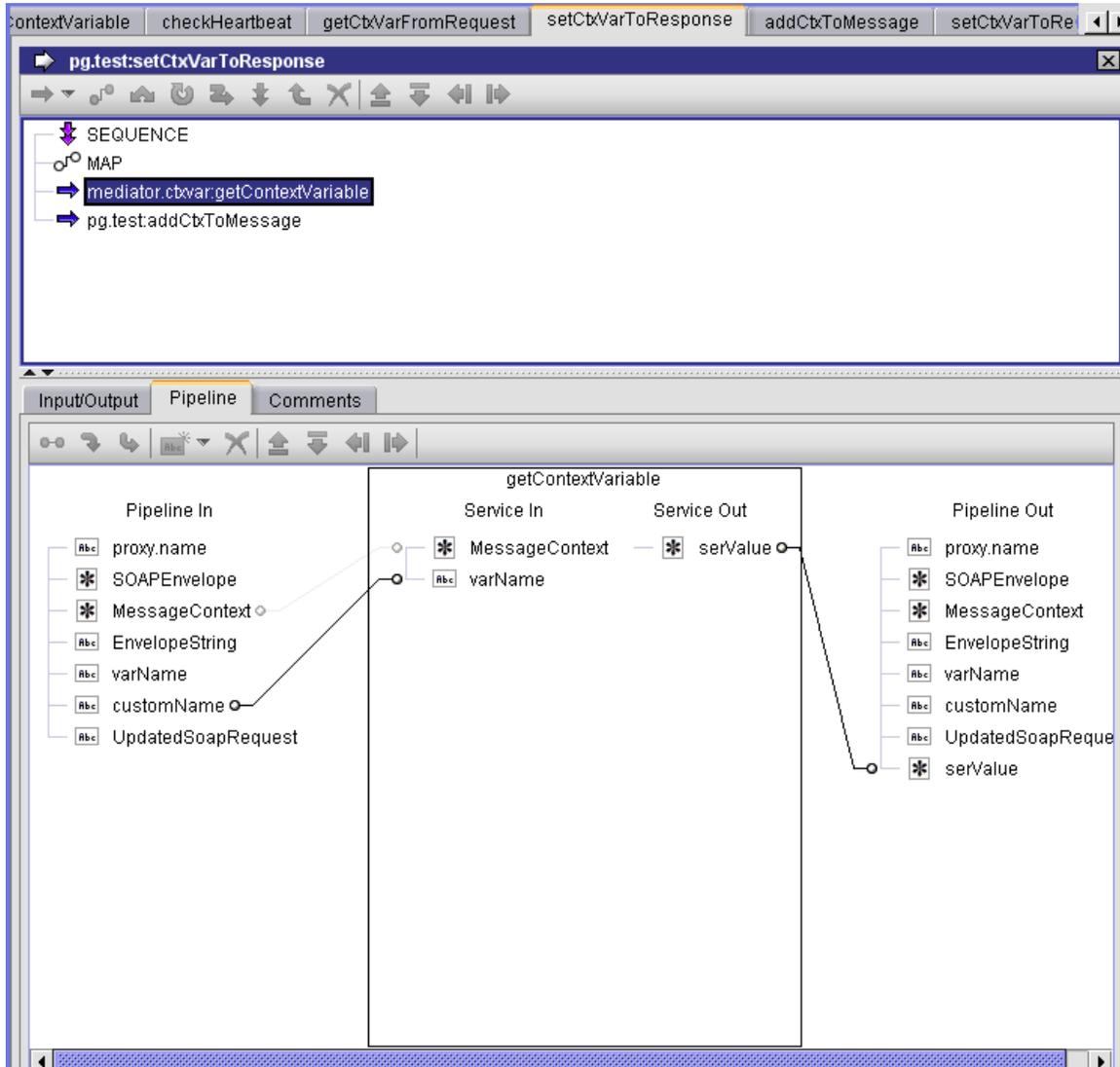
Clicking on the `customName` pipeline variable will display the name.

Step 3. Displaying the value of customName

The screenshot displays a software interface for configuring a pipeline. At the top, a tab bar shows several tabs: `ontextVariable`, `checkHeartbeat`, `getCtxVarFromRequest`, `setCtxVarToResponse` (selected), `addCtxToMessage`, and `setCtxVarToRe`. Below the tabs, the main window title is `pg.test:setCtxVarToResponse`. The pipeline structure is shown in a tree view on the left, including a `SEQUENCE` block containing a `MAP` block, followed by `mediator.ctxvar:getContextVariable` and `pg.test:addCtxToMessage`. An `Input/Output` panel is visible on the left, listing various input and output variables. A dialog box titled `Input for 'customName'` is open in the center, showing the `customName` field with the value `mx:COMP_TEST`. The dialog also includes checkboxes for `Overwrite pipeline value` (checked) and `Perform variable substitution` (unchecked). The `Pipeline Out` section on the right shows the output structure, including `proxy.name`, `SOAPEnvelope`, `MessageContext`, `EnvelopeString`, `varName`, `customName` (highlighted with a blue selection bar), and `UpdatedSoapRequest`.

The call to `pub.mediator.ctxvar:getContextVariable` retrieves the value of the custom context variable from the context variable map.

Step 4. Calling mediator.ctxvar:getContextVariable



This is just a sample Java service that takes the context variable and creates a top-level element in the response message using the same name and value.

Step 5. Sample service using the context variable

The screenshot displays a software interface for configuring a pipeline. At the top, a tabbed menu includes: contextVariable, checkHeartbeat, getCtxVarFromRequest, setCtxVarToResponse, addCtxToMessage, and setCtxVarToRe. The active window is titled "pg.test:setCtxVarToResponse".

The main area shows a tree view of the pipeline structure:

- SEQUENCE
 - MAP
 - mediator.ctxvar:getContextVariable
 - pg.test:addCtxToMessage

9 Important Considerations when Configuring SOAP-based Virtual Services

■ Handling Services with Multiple Ports and Bindings	296
--	-----

Handling Services with Multiple Ports and Bindings

Mediator implicitly assumes that there is a one-to-one mapping between a WSDL service and a Virtual Service. A problem arises if a `<service>` element contains multiple `<port>` elements that point to different bindings (and consequently different port types) -- the problem is that Mediator will create a virtual service that has the operations from only *one* of the `portTypes`. Mediator chooses the first port available under `<service>` and exposes the operations corresponding to the equivalent binding/portType.

For example, consider the following WSDL fragment that shows the structure of the `portType`, `binding` and `service` elements in the WSDL. Note that there are:

- Two distinct `<portType>` elements: `SystemPortType` and `CustomerPortType`.
- Two equivalent bindings defined for each `<portType>`: `SystemBinding` and `CustomerBinding`.
- A single `<service>` element that defines the two ports with distinct endpoints (one for each binding available).

Example `<portType>` Elements

```
<portType name="SystemPortType">
  <operation name="ping">
    ...
  </operation>
</portType>
<portType name="CustomerPortType">
  <operation name="getOperation1">
    ...
  </operation>
  <operation name="getOperation2">
    ...
  </operation>
  <operation name="getOperation3">
    ...
  </operation>
  <operation name="getOperation4">
    ...
  </operation>
</portType>
```

Example `<binding>` Elements

```
<binding name="SystemBinding" type="tns:SystemPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="ping">
    ...
  </operation>
</binding>
<binding name="CustomerBinding" type="tns:CustomerPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getOperation1">
    ...
  </operation>
  <operation name="getOperation2">
    ...
  </operation>
```

```

</operation>
<operation name="getOperation3">
  ...
</operation>
<operation name="getOperation4">
  <soap:operation soapAction="urn:customer.service.cm.be/getOperation4"/>
  ...
</operation>
</binding>

```

Example <service> Element

```

<service name="CustomerRefService">
  <port name="SystemPort" binding="tns:SystemBinding">
    <soap:address location="http://.../v4/SystemPort"/>
  </port>
  <port name="CustomerPort" binding="tns:CustomerBinding">
    <soap:address location="http://.../v4/CustomerRefPort"/>
  </port>
</service>

```

There are two options for a workaround to this problem.

Workaround Option 1

You can create two different virtual services. That is, you can expose this native service as two virtual services -- one for each operation that needs to be invoked:

- A service for the `getXXX` operations (for example, a service called `VS_Customer`).
- A service for the `ping` operation (for example, a service called `VS_ping`).

To create a different virtual service for each operation

1. Create a virtual service for the native service `CustomerRefService` and name it `VS_Customer`, for example.
2. Configure `VS_Customer` and configure its routing protocol as "Straight Through".
3. Specify the routing address as `http://... /v4/CustomerRefPort` (for `CustomerBinding`, where all the `getXXX` operations are supported).
4. On the virtual service's **Summary** profile, click on the URL for the WSDL (this a copy of the virtual service template WSDL, very similar to the original native service WSDL) and download/save it to your local file system.

(Make an additional copy of this downloaded WSDL in case you make a mistake in your editing.)

5. Remove the following elements from the WSDL and then save it:
 - `SystemPortType`
 - `SystemBinding`
 - `SystemPort`

Note: Make sure your browser or XML tool can read this modified WSDL without any syntax error.

6. Attach the modified WSDL file to the virtual service by selecting the **Attach WSDL** command in the virtual service's **Actions** menu.
7. Create another virtual service for the native service `CustomerRefService` and name it `VS_ping`, for example. Repeat the above steps but with the following differences:
 - Specify the routing address as `http://... /v4/SystemPort`.
 - Remove the following elements from the WSDL:
 - `CustomerPortType`
 - `CustomerBinding`
 - `CustomerPort`
8. Deploy both virtual services to Mediator.

Workaround Option 2

With this option, you expose the native service as one virtual service. The web service client will access the service through one address to the virtual service for all the possible operations (`ping` and `getXXX`). The virtual service then takes care of routing to the correct endpoint for the different operations. This is accomplished by using "Content-based" routing (instead of "Straight Through" routing) to determine the operation being called (based on the SOAP request content) and then forwarding the request to the correct endpoint.

To create a virtual service with "Content-based" routing

1. Create a virtual service for the native service `CustomerRefService` and name it `VS_CustomerRefService`, for example.
2. Configure `VS_CustomerRefService` and configure its routing protocol as "Content-based".
3. On the **Routing Protocols** tab, construct the routing rule as follows:
 - a. Click the **Endpoint** button (next to the **Route To** column).
 - b. In the **Search for Endpoint** dialog that appears, click the **Search** button to search for the Web service endpoint to route the requests to.
 - c. Select `http://... /v4/SystemPort` (the Accessing URI that goes to ping operation).
 - d. To create an XPath expression for the routing rule, click the **Edit** button (next to the **If True** column).
 - e. In the XPath Editor that appears, click the **All Nodes** tab, expand the namespace's node, click to highlight the `tns:ping` element, and click **OK**

- f. Double check that you have something like this in the rule and it is routed to `SystemPort`:

```
/soapenv:Envelope/soapenv:Body/tns:pingRequest
```

4. Set the **Default To** routing field to the routing address `http://... /v4/ CustomerRefPort` (for `CustomerBinding`, where `getXXX` operations are supported).
5. On the virtual service's **Summary** profile, click on the URL for the WSDL (this a copy of the virtual service template WSDL, very similar to the original native service WSDL) and download/save it to your local file system.

(Make an additional copy of this downloaded WSDL in case you make a mistake in your editing.)

6. Modify the WSDL as follows and then save it:
 - a. Copy the `ping` operation from `SystemPortType` and add into `CustomerPortType`.
 - b. Delete the `SystemPortType`. The objective here is to make *one* port type only.
 - c. Update the `SystemBinding` to also refer to `CustomerPortType`, since `SystemPortType` has been deleted.

Note: The `soapAction` attribute must be specified for the `soap:operation` element to ensure that Mediator can resolve the operation being invoked for this service.

- d. Save the WSDL.

Note: Make sure your browser or XML tool can read this modified WSDL without any syntax error.

7. Attach the modified WSDL file to the virtual service by selecting the **Attach WSDL** command in the virtual service's **Actions** menu.
8. Deploy the virtual service to Mediator.

10 Important Considerations when Configuring REST or XML Virtual Services

■ Introduction	302
■ Endpoint Manipulation of REST or XML Virtual Services	302
■ The Request Message's HTTP Methods and Content-Types for REST and XML Services	304
■ Changing the HTTP Method of a REST or XML Request	305
■ Working with the JSON Content-Type	310
■ Handling Virtual REST APIs with Multiple Resources	321

Introduction

With Web services, the native service endpoint that is sent by CentraSite to Mediator inside a virtual service definition is a static element. Once the virtual service is successfully deployed to Mediator, the real endpoint is returned to CentraSite as part of the response message during deployment. At run time, when a SOAP request is received, that request is POSTed to the endpoint that is statically defined in the virtual service definition.

However, with REST services or XML services, the endpoint is flexible. The REST services or XML services describe data as resources. The resources are accessible via logical endpoints that have application meaning to users. For example, a collection of textbooks might be defined as a resource with the following URL:

```
http://{host}:{port}/books
```

A specific book with an identifier of 1234 would be accessible with the following URL:

```
http://{host}:{port}/books/1234
```

Due to this difference and others, you should keep the following topics described in this chapter in mind when you configure a REST or XML virtual service.

Endpoint Manipulation of REST or XML Virtual Services

When you configure a virtual REST or XML service, you specify the native service name, an endpoint, and the HTTP method type(s) that are included in the message (POST, GET, PUT, DELETE). From this information, CentraSite will generate a virtual service definition that includes service and operation elements, as well as an endpoint and binding element pair for each HTTP method specified.

CentraSite will automatically generate an operation name to be included when the virtual service definition deployment message is sent to Mediator. This means that if you create a virtual service called VS1, and you specify a native endpoint, then the endpoint exposed by Mediator for calling the virtual service will be /ws/VS1/Invoke.

For example, assume the following endpoints are deployed:

- Native service endpoint: `http://localhost:8080/services/mtc/member`
- Virtual service endpoint: `http://localhost:5555/ws/VS1/Invoke`

Assume that the example virtual service is deployed with two HTTP method bindings: GET and POST. Both of these bindings have operation elements that include the same HTTP location attribute: member. To better illustrate the functionality, the examples below show a series of sample requests from a consumer including the requests' HTTP method and Content-Type. (At run time, REST message detection is dependent upon a consumer using the correct Content-Type when a request is sent.) Each example shows the expected endpoint that Mediator will send after it has rewritten the endpoint prior to native service invocation.

Example 1

For a GET, assume that:

The request Content-Type is: application/x-www-form-urlencoded

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member

The application function is: The native service returns a collection of members with summary information.

Example 2

For a GET, assume that:

The request Content-Type is: application/x-www-form-urlencoded

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke/1234

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member/1234

The application function is: The native service returns summary data for a member with this key.

Example 3

For a GET, assume that:

The request Content-Type is: application/x-www-form-urlencoded

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke/1234?detail=true

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member/1234?detail=true

The application function is: Query parameters remain intact. Returns a response message with more member details.

Example 4

For a POST, assume that:

The request Content-Type is: application/xml or application/json

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke/1234

The native service endpoint sent by Mediator is: `http://localhost:8080/services/mtc/member/1234`

The application function is: The request message provides the contents needed to create the member resource.

Example 5

For a GET, assume that:

The request Content-Type is: `application/x-www-form-urlencoded`

The endpoint received by Mediator is: `http://localhost:5555/ws/VS1/Invoke/joe`

The native service endpoint sent by Mediator is: `http://localhost:8080/services/mtc/member/joe`

The application function is: Fetches the member defined with the login joe. (Mediator contains no metadata in its service deployment to differentiate between the "login" vs. "key" GET requests.)

Example 6

For a GET, assume that:

The request Content-Type is: `application/x-www-form-urlencoded`

The endpoint received by Mediator is: `http://localhost:5555/ws/VS1/Invoke?type=login&value=joe`

The native service endpoint sent by Mediator is: `http://localhost:8080/services/mtc/member?type=login&value=joe`

The application function is: The native service might also support a static endpoint with constraints defined in query parameters. Mediator also supports this approach.

The Request Message's HTTP Methods and Content-Types for REST and XML Services

When you configure the Entry Protocol step of a virtual REST or XML service, it is important to specify all the HTTP methods that are supported for the service. For example, if the virtual service is deployed to Mediator and you selected only the GET method in the virtual service's details page, then Mediator will only permit GET invocations. In this case, a POST request will be rejected with a return of statusCode 405 even if the native service happens to support POSTs.

It is important that the client's requests contain an HTTP Content-Type header. At run time, Mediator determines which message builder to use based on the message's HTTP

method and its Content-Type. (The absence of the soapAction header will indicate to Mediator that the message is an XML message.)

The valid HTTP method/Content-Type combinations are as follows:

This method...	Can be included in a message of this Content-Type...
POST	application/xml application/json application/x-www-form-urlencoded multipart/form-data or text/xml
PUT	application/xml application/json application/x-www-form-urlencoded multipart/form-data or text/xml
GET	application/x-www-form-urlencoded
DELETE	application/x-www-form-urlencoded
Note:	<p>Keep the following points in mind:</p> <ul style="list-style-type: none"> ■ If Mediator receives a request sent with an HTTP method that is not specified in the virtual REST service or virtual XML service definition, it will return a 405 error. ■ If Mediator receives a request sent with a wrong Content-Type, it will return a 415 error. In addition, if the wrong Content-Type is used with a GET or DELETE, then the query parameters contained in the message (if any) will not be processed.

Changing the HTTP Method of a REST or XML Request

When configuring a REST or XML virtual service, you specify whether to route the requests to the native service with the same HTTP method that is contained in the requests (GET, POST, PUT, DELETE), or whether to route the requests with a different HTTP method.

Typically you want to pass each request to the native service with the same HTTP method that is contained in the request. For example, if a request contains a GET method, you allow the GET method to be passed to the native service. However, there might be rare cases in which you want to change the HTTP method of a request to different HTTP method. For example, you might want to:

- Expose an XML service as a REST service.

In this case, the service you create would be a Virtual XML service that exposes the HTTP methods GET, POST, PUT, and DELETE, but the routing method would always be POST.

- Expose a REST service whose virtual REST service only exposes the POST method.

To change the HTTP method of a REST or XML request

- On the REST or XML virtual service, set the value of the HTTP Method field either statically (by explicitly setting the value to **GET**, **POST**, **PUT**, or **DELETE**) or dynamically (by setting the value to **Use Context Variable**).

In order to use the **Use Context Variable** option to set the field dynamically, you must write a webMethods IS service that sets a value of GET, POST, PUT or DELETE for a predefined context variable named ROUTING_METHOD. You need to invoke this service in the virtual service's **Invoke webMethods IS Service** action. For details, see "[Changing HTTP Methods in Requests Dynamically using a Context Variable](#)" on page 308.

Caution: Use this feature carefully, since changing HTTP methods to certain other HTTP methods could result in unintended results or errors.

For example, changing an inbound GET request to a DELETE request would be a serious mistake if the deletion was not intended and the native REST service actually deleted a resource when invoked with a DELETE method. Additionally, an incoming POST or PUT request cannot be translated into a GET or DELETE if the request has nested elements. For more information, see "[The Implications of Changing HTTP Methods](#)" on page 306.

The Implications of Changing HTTP Methods

When changing this incoming HTTP method...	To...	Note that...
GET	POST	<ul style="list-style-type: none"> ■ The Content-Type of the changed request is sent as application/xml or application/json, and the charset is UTF-8. ■ Depending on the structure of the native service, be aware that the native service might not be expecting the same payload structure that is being sent. In this case, you would need to transform the request message into the format required by the native service before Mediator sends the requests to the native service. For more information, see "Sample"

When changing this incoming HTTP method...	To...	Note that...
XSLT Transformation for GET-to-POST or GET-to-PUT" on page 309.		
GET	PUT	Identical to GET-to-POST, except that Mediator changes the request's HTTP method from GET to PUT.
GET	DELETE	No comment.
POST	GET	<ul style="list-style-type: none"> ■ Mediator will translate the POSTed request elements into query string parameters, in a root element. <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p>Note:An incoming POST or PUT request cannot be translated into a GET or DELETE if the request has nested elements. For example:</p> <pre style="font-family: monospace; font-size: 0.9em;"> (this is correct) <person> <lastName>Smith</lastName> </person> (this is incorrect) <person> <name> <last>Smith</last> </name> </person> </pre> </div> <ul style="list-style-type: none"> ■ If you want to send additional parameters as part of the request URL, you can transform this payload. To do this, you can use an XSLT file or a webMethods IS service call to add parameters before the request is sent to the native service. For more information about adding parameters to REST virtual services, see <i>Working with REST-based APIs in CentraSite</i>.
POST	DELETE	Identical to POST-to-GET, except that Mediator changes the request's HTTP method from POST to DELETE.
POST	PUT	The Content-Type of the changed request is sent as application/xml or application/json, and the charset is UTF-8.

When changing this incoming HTTP method...	To...	Note that...
PUT	GET	Identical to POST-to-GET, except that Mediator changes the request's HTTP method from PUT to GET.
PUT	POST	The Content-Type of the changed request is sent as application/xml or application/json, and the charset is UTF-8.
PUT	DELETE	Identical to POST-to-DELETE, except that Mediator changes the request's HTTP method from PUT to DELETE.
DELETE	GET	No comment.
DELETE	POST	Identical to GET-to-POST, except that Mediator changes the request's HTTP method from DELETE to POST.
DELETE	PUT	Identical to GET-to-PUT, except that Mediator changes the request's HTTP method from DELETE to PUT.
GET, POST, PUT or DELETE	Use Context Variable	See "Changing HTTP Methods in Requests Dynamically using a Context Variable" on page 308.
GET or DELETE	POST or PUT	Note that the query parameters will be picked off the URL and stored as top-level elements when the message is sent to the native service. The query parameters are ignored on the endpoint URL and lost when we POST to the native provider (that is, don't change the protocol method).

Changing HTTP Methods in Requests Dynamically using a Context Variable

Alternatively, instead of changing an HTTP method explicitly (statically) to PUT, POST, GET or DELETE, you can change the HTTP method to the value of a predefined context

variable (ROUTING_METHOD) that dynamically resolves to a different HTTP method (PUT, POST, GET or DELETE, as appropriate).

To change the HTTP method dynamically, you create a webMethods IS service and invoke it in the virtual service's **Invoke webMethods IS Service** action. This webMethods IS service should reference the predefined context variable ROUTING_METHOD (see "[The Predefined Context Variables](#)" on page 274). To set the value of ROUTING_METHOD, use the setContextVariableValue method, which is defined in the following class:

```
com/softwareag/mediator/api/MediatorRuntimeFacade.java
```

For example:

```
public static final void updateHttpMethod(IData pipeline)
    throws ServiceException {

    String mcKey = "Message Context";

    Object obj = IDataUtil.get(pipeline.getCursor(), mcKey);
    if (obj!=null && obj instanceof
org.apache.axis2.context.MessageContext) {

        MessageContext msgCtx = (MessageContext) obj;

        QName varName =
new QName(MediatorContextVariableType.ROUTING_METHOD.getName());

        MediatorRuntimeFacade.setContextVariableValue(varName, "PUT", msgCtx );
    }
}
```

Sample XSLT Transformation for GET-to-POST or GET-to-PUT

As stated in the above table, depending on the structure of the native service, the native service might not be expecting the same payload structure that is being sent. In this case, you would need to transform the request message into the format required by the native service before Mediator sends the requests to the native service. To do this, you invoke an XSLT file at run time.

Assume that:

- The native service name is "authors".
- The virtual REST service or virtual XML service for "authors" is named "vs-authors" and is made available in Mediator at this endpoint: `http://localhost:5555/ws/vs-authors/Invoke`. The targetNamespace of the virtual REST service or virtual XML service is "`http://example.com/authors`".

Following is a sample XSLT transformation file for the GET-to-POST or GET-to-PUT scenario.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ns="http://example.com/authors"
    version="1.0">

    <xsl:output method="xml" omit-xml-declaration="no" standalone="yes"
        indent="yes"/>
```

```

<xsl:strip-space elements="*" />
<xsl:template match="node()|@*">
  <xsl:copy>
    <xsl:apply-templates select="node()|@*" />
  </xsl:copy>
</xsl:template>

<xsl:template match="//ns:invoke/node()">
  <xsl:element name="{local-name(.)}">
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>

<xsl:template match="//ns:invoke">
  <xsl:element name="authors">
    <xsl:apply-templates />
  </xsl:element>
</xsl:template>
</xsl:stylesheet>

```

Working with the JSON Content-Type

Mediator can accept a REST service request that specifies the Content-Type "application/json" (or "application/json/badgerfish") and the HTTP methods PUT, GET, DELETE, and POST.

Assuming that the native service supports both JSON and the HTTP method(s) specified in the request, Mediator can determine the correct service, operation and output format (JSON) to return to the consuming application. There are different ways in which a native service provider can be prompted to return response content. It will vary with the provider. For example, some providers may rely on the Accept transport header to specify the format the consumer wants. Others may use an element in the request or a query parameter on the URL.

However, suppose for example that the native service does not support the HTTP method specified in the request (for example, POST). As a workaround, you can configure the virtual service so that it "bridges" this difference between the consumer request and the native service. In this case, you can configure the virtual service so that it takes the POST and bridges it into an HTTP GET query, and then returns the service to the consumer in the expected JSON format. To implement this, you set the following predefined context variables in a user-defined webMethods IS service that you can invoke in the virtual service's **Invoke webMethods IS Service** action:

- **MESSAGE_TYPE**: A Content-Type defined in axis2.xml for a message formatter. This value must be a key in the axis2 message formatters list, since it is used to control message serialization. (The valid choices are defined as attributes of the <messageFormatters/> group in the Integration Server's axis2.xml configuration.)
- **BUILDER_TYPE**: A Content-Type defined in axis2.xml for a message builder. This value must be a key in the axis2 message builders list since it is used to control building of native service response messages. (The valid choices are defined as attributes of the <messageBuilders/> group in the Integration Server's axis2.xml configuration.)

This and other "bridging" scenarios are discussed in this section.

How Mediator Determines Which Builder and Formatter Classes to Use (and How You Can Override Them)

Mediator makes these determinations at run time as follows. This table also summarizes how you can override the default determinations.

Run-time Step	Description
Mediator receives the request from the client	It is important that the client's PUT or POST requests contain the HTTP header Content-Type because the Content-Type header determines the message builder Mediator uses to parse the input stream. (GET or DELETE request do not require a Content-Type header.)
Mediator sends the request to the service provider	<p>Mediator uses a message formatter to serialize the request, and then sends the serialized request to the native service provider.</p> <p>Mediator determines the message formatter to use as follows:</p> <ul style="list-style-type: none"> ■ If you explicitly specify a message formatter (by setting the MESSAGE_TYPE context variable in a webMethods IS service and invoking this service in the virtual service's Invoke webMethods IS Service action), then Mediator uses that formatter. The Content-Type header that Mediator sends to the native provider is the one that is associated with the MESSAGE_TYPE context variable. ■ Else, Mediator will use the message formatter associated with the Content-Type sent by the client (and sends the Content-Type to the native provider). ■ Else, if no Content-Type was sent by the client, then: <ul style="list-style-type: none"> ■ For PUT requests, the default formatter used (and the Content-Type header that Mediator sends to the native provider) is <code>application/xml</code>. ■ For POST requests, the default formatter used (and the Content-Type header that Mediator sends to the native provider) is <code>SOAP</code> (and the request will fail). ■ For GET or DELETE requests (which do not require a Content-Type header), the default

Run-time Step	Description
	formatter used (and the Content-Type header that Mediator sends to the native provider) is <code>application/x-www-form-urlencoded</code> .
Mediator receives a response from the service provider	<p>When the provider returns a response to Mediator, a message builder parses the response stream into an Axiom message to be stored in the message context. Mediator determines which message builder to use as follows:</p> <ol style="list-style-type: none"> 1. Mediator will select the message builder associated with the request's Accept transport header, if one was specified. 2. Else, you can set the <code>BUILDER_TYPE</code> context variable in a webMethods IS service, and invoke this service in the virtual service's Invoke webMethods IS Service action. Mediator will check that the builder type is a valid Content-Type for the list of builders in <code>axis2.xml</code>. This variable takes priority over the current setting specified in the Accept transport header. That is, Mediator will only use the Accept header to determine the builder type needed to parse a native provider response if no IS service was written to set the <code>BUILDER_TYPE</code> context variable. 3. Else, Mediator will use the builder associated with the Content-Type specified in the request (assuming that the Content-Type is one of the types that is mapped in <code>axis2.xml</code>). 4. Else, if no Content-Type was specified in the request (for example, a PUT or POST request with no Content-Type, or any GET or DELETE request), or if the Content-Type is not one of the types that is mapped in <code>axis2.xml</code>, then Mediator will default to <code>application/xml</code>.
Mediator sends a response to the client	<p>Mediator serializes the response and sends it to the client.</p> <p>By default, Mediator uses the formatter that was used to serialize the request sent to the provider. (If the formatter was <code>application/x-www-form-urlencoded</code> (for a GET or DELETE request), then Mediator will instead use <code>application/xml</code> so it can send the response.)</p>

Run-time Step	Description
	You can override the Content-Type that is sent to the client by setting the MESSAGE_TYPE context variable in a webMethods IS service, and invoking this service in the virtual service's Invoke webMethods IS Service action.

Scenarios for Requesting JSON Type Services

Following are some of the possible scenarios in which JSON type services can be requested. Many scenarios require that you "bridge" the differences between consumer requests and the native service (that is, differing HTTP methods and Content-Types). Three of the scenarios are discussed in more detail following the table.

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging?
GET	GET	JSON	JSON	Request Processing step bridging (see "JSON Example 1: GET Request, JSON Response" on page 315)
POST/JSON	POST/JSON	JSON	JSON	No bridging needed
GET	GET	XML	JSON	Response Processing step bridging
POST/JSON	GET	JSON	JSON	Request Processing step bridging

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging?
POST/ JSON	GET	XML	JSON	Request Processing step bridging and Response Processing step bridging (see "JSON Example 2: POST/JSON Request, JSON Response (where POST is not supported)" on page 316)
POST/ JSON	XSLT/GET *	JSON	JSON	Request Processing step bridging
POST/ JSON	XSLT/POST/ XML *	XML	XML	Request Processing step bridging
POST/ JSON	POST/JSON	JSON/ XSLT *	JSON	Response Processing step bridging
GET	GET	JSON	XML	Request Processing step bridging and Response

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging?
				Processing step bridging (see "JSON Example 3: GET Request, XML Response" on page 318)
POST/ XML	POST/JSON	JSON	JSON	Request Processing step bridging

* The XSLT references indicate where you can perform an XSLT message transformation at either the **Request Transformation** or **Response Transformation** action.

In the table above, the required Content-Type settings are not shown, but assume the following:

HTTP Method/Request Content	Required Axis2 Content Type
GET or DELETE	application/x-www-form-urlencoded
POST or PUT/XML	application/xml
POST or PUT/Mapped JSON	application/json
POST or PUT/Badgerfish	application/json/badgerfish

JSON Example 1: GET Request, JSON Response

In this example, a consumer sends a GET request to get a native JSON service. Mediator will send the response to the consumer in the requested JSON format (as indicated by the "output=json" parameter in the query).

The request looks like this:

`http://localhost:5555/ws/YahooVS/search?query=wsdl20&output=json`

and because this is a GET request, the Content-Type defaults to `application/x-www-form-urlencoded`.

Note: For GET or DELETE requests for REST services, it is not necessary to specify the Content-Type in the request; Mediator will default to `application/x-www-form-urlencoded` for GET or DELETE requests.

The run-time processing will be as follows:

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging
GET	GET	JSON	JSON	Request Processing step bridging

Since the request is a GET (that is, of Content-Type `application/x-www-form-urlencoded`), but Mediator expects to receive a JSON stream from the provider, you must send the `BUILDER_TYPE` `application/json` to the native provider. To do this, write and invoke a webMethods IS service in the virtual service's **Invoke webMethods IS Service** action. The IS service should include the following predefined context variable set to this value:

Context Variable	Value
<code>BUILDER_TYPE</code>	<code>application/json</code>

JSON Example 2: POST/JSON Request, JSON Response (where POST is not supported)

In this example, a consumer sends a POST request (of Content-Type `application/json`) to a native service, but the native service does not support POST.

The request's Content-Type is `application/json` and its output parameter is set to `xml`. The reason for this is explained below.

The run-time processing will be as follows:

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging
POST/JSON	GET	XML	JSON	<ul style="list-style-type: none"> ■ Request Processing step bridging ■ Response Processing step bridging

Configure the virtual service as follows:

- In the virtual service's Routing Protocols tab, set the value of the HTTP Method field to the value GET. Doing this instructs Mediator to change the POST to an HTTP GET before Mediator sends it to the native service (which is necessary because the native service does not support POST).
- Write and invoke a webMethods IS service in the virtual service's **Invoke webMethods IS Service** action. The IS service should include the following predefined context variables set to the values shown below. The request's "query": "wsdl120" and "output": "xml" parameters are transformed into query parameters on the URL before the native service is invoked. Thus, although the consumer is sending a JSON request, the native service is instructed to return an XML response to Mediator.

Context Variable	Value
MESSAGE_TYPE	application/x-www-form-urlencoded
BUILDER_TYPE	application/xml

- Write and invoke a webMethods IS service in the virtual service's **Invoke webMethods IS Service** action. The IS service should include the following predefined context variable set to the value shown below. Doing this instructs Mediator to bridge the XML response from the native service into JSON format, to be returned to the consumer.

Context Variable	Value
MESSAGE_TYPE	application/json

JSON Example 3: GET Request, XML Response

In this example, a consumer sends a GET request to get a native service. Mediator will send the response to the consumer in the requested XML format (as indicated by the "output=xml" parameter in the query).

The request looks like this:

```
http://localhost:5555/ws/YahooVS/search?query=wsdl20&output=xml
```

and because this is a GET request, the Content-Type defaults to `application/x-www-form-urlencoded`.

The run-time processing will be as follows:

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging
GET	GET	JSON	XML	<ul style="list-style-type: none"> ■ Request Processing step bridging ■ Response Processing step bridging

Since the request is a GET (that is, of Content-Type `application/x-www-form-urlencoded`), but Mediator expects to receive a JSON stream from the provider, you must instruct Mediator to send the `BUILDER_TYPE` `application/json` to the native provider. To do this, write and invoke a webMethods IS service in the virtual service's **Invoke webMethods IS Service** action. The IS service should include the following predefined context variable set to this value.

Context Variable	Value
<code>BUILDER_TYPE</code>	<code>application/json</code>

Since the provider will return a JSON stream to Mediator, but the consumer expects to receive the service in XML output format, you must set the `MESSAGE_TYPE` to `application/xml`. To do this, write and invoke a webMethods IS service in the virtual service's **Invoke webMethods IS Service** action. The IS service should include the following predefined context variable set to this value:

Context Variable	Value
MESSAGE_TYPE	application/xml

Characteristics of the Mapped and Badgerfish JSON Conventions

The open source library that Axis2 uses to support JSON is called Jettison. The Jettison library supports two formats of JSON: Mapped JSON and Badgerfish. Both are syntactically correct from a JSON perspective.

Note: An important difference between the two is that the Mapped convention returns a service fault if a virtual service is configured for application/json (Mapped convention) and it encounters a message that has namespaces, while the Badgerfish convention attempts to avoid losing any meaning encoded in XML by preserving namespace declarations. The Axis2 JSON library MessageFormatter will complain if Mediator attempts to transform an XML response that contains namespace declarations. So, either make sure that the requests do not include namespaces, or else set the MESSAGE_TYPE to application/json/badgerfish instead of setting it to application/json.

Other characteristics include the following.

Mapped JSON Convention

1. An element with no characters or child elements is represented by:

```
{ "element" : "" }
```

2. No namespaces declarations are ever written.

Note: The Badgerfish convention does allow namespaces. If a client sends a request that contains XML namespaces, you need to bridge to the Badgerfish convention. To do this, in the virtual service's **Set Headers** action, set the parameter **Set Headers** to **Header** and specify the Name as Content-Type and the Value as application/json/badgerfish. Doing this will override the existing Content-Type that will be sent to the native provider.

3. An element with multiple child elements of the same name is represented by an array.

Simple case:

```
<price>10.00</price>
{ "acme.price" : { "10.00" } }
```

Array case:

```
<root><child>test</child><child>test</child></root>
{ "root" : { "child" : [ "test", "test" ] } }
```

4. The XML attributes for a message are prefixed with @ when a message is serialized (same as Badgerfish).

Badgerfish Convention

This convention is used to provide the means to translate between XML and JSON without losing any data (that is, namespaces).

1. Element names become object properties.
2. Text content of elements goes in the \$ property of an object.
3. Nested elements become nested properties.
4. Multiple elements at the same level become array elements.
5. Attributes go in properties whose names begin with @.
6. Active namespaces for an element go in the element's @xmlns property.
7. The default namespace URI goes in @xmlns.\$.
8. Other namespaces go in other properties of @xmlns.
9. Elements with namespace prefixes become object properties, too.

Simple example:

```
<price xmlns="http://acme.com">10.00</price>
{ "price": { "@xmlns": { "$": "http://acme.com" }, "$1": "10.00" } }
```

A more complex example:

```
<alice xmlns="http://some-namespace"
  xmlns:charlie="http://another-namespace">
  <bob>david</bob>
  <charlie:edgar>frank</charlie:edgar>
</alice>

{ "alice" : { "bob" : { "$" : "david" , "@xmlns" :
{"charlie" : "http://another-namespace" , "$" : "http://some-namespace" } } ,
  "charlie:edgar" : { "$" : "frank" , "@xmlns" :
{"charlie":"http://another-namespace", "$" : "http://some-namespace" } },
"@xmlns" : { "charlie" : "http://another-namespace", "$" : "http://some-namespace" } } }
```

Multiple Root Nodes in JSON REST Services

With REST virtual services, when working with requests and responses of the Content-Type application/json, the message content can contain one or more root nodes.

For example, a message might have the two root nodes {"firstName": "John", "lastName": "Smith"}. Note the following points for messages with multiple root nodes.

- The XSLT provided in the Request/Response Processing step should have the XPath start with //, for example //firstName. This is because during the processing of the JSON content, Mediator will wrap the given content with system-defined elements.

- webMethods IS services that you invoke in the Request/Response Processing step will contain the JSON content in the variable called `JSONRESTContentString`. You can update the content in the IS service and put the updated content into the pipeline's input variable `UpdatedJSONRESTContentString`, which will be sent to the native service.

With REST virtual APIs, if you have the XSLT or webMethods IS services defined in the Request/Response Handling actions, then you must have the method name prefixed to the resource name in order to reference to that particular resource. For example, if you have the resource name `phones` defined with a `GET` method, you would reference it as `GET_phones`.

Handling Virtual REST APIs with Multiple Resources

The enhanced REST framework of CentraSite Business UI allows you to explicitly define multiple resources for a RESTful API. Each resource within the API exposes a unique URI for performing the CRUD operations on the resource.

CentraSite provides the resource path tokenizer to support the multiple resources handling at run-time. The tokenizer for substituting a resource path is automatically appended to the API's endpoint (base URL) by a path variable `${sys:resource_path}`. When there are multiple resources defined for an API, at run time, Mediator replaces the resource path tokenizer with the appropriate resource path that is defined for the particular resource call.

Important: Beginning with version 9.7, CentraSite supports the multi-resource handling of the REST interface at run-time (in contrast, earlier versions of CentraSite supported a single resource handling). Note that the enhanced REST interface that is implemented by current version of CentraSite is not compatible with the interface that was implemented by previous versions of Mediator prior to version 9.7. However, if you still attempt to publish a virtual REST API with multiple resources to an earlier version of Mediator instance, Mediator throws an exception message.

Let's now learn about the usage of resource path tokenizers in CentraSite by looking at a couple of examples using our sample phone store API.

Consider you have a plain REST API `PhoneStoreAPI` with a defined set of resources for performing the CRUD operations.

Assume you have the `PhoneStore` API with the following configuration details:

- Base URL:

```
http://www.phonestore.com/api
```

- Resource:

```
phones
```

- Resource URI:

```
/phones
```

- **Native Endpoint:**

```
http://www.phonestore.com/api/phones
```

Beginning with version 9.7, on a virtual copy of this API, CentraSite appends the resource path tokenizer `${sys:resource_path}` to represent its Route to endpoint like this:

```
http://www.phonestore.com/api/${sys:resource_path}
```

In the aspect of this resource path tokenizer, there are two principal scenarios, when consumers attempt to model virtual REST APIs with multiple resources.

Scenario A

Consider you have a virtual REST API created using the current version of CentraSite, this API will have the Route to endpoint of the Straight-Through/Content-Based/Content-Based Routing action, by default, appended with a resource path tokenizer. The Route to endpoint assigned by CentraSite has the format `<base-url>/${sys:resource_path}`, where the `resource_path` is the same as the original resource URI defined in the API's first resource definition.

Whenever you try to add a new resource to this API, the endpoint is automatically assigned and sent to Mediator for processing inside a Virtual Service Definition (VSD) in the prescribed format. At run time, when a HTTP request is received for the resource, that request will be sent to the endpoint dynamically substituted with the path variable. This dynamic substitution of the endpoint indicates that the REST interface is enhanced to support multiple resources.

Scenario B

Consider you have a virtual REST API created in versions of CentraSite prior to 9.7, this API will continue to exhibit the old REST behavior, that is, it will continue to send the HTTP requests to the native API endpoint using the resource URI defined in the earlier version.

Now consider our sample PhoneStore API with two different formats of resource URIs, say, `/phones`, `/invoke`

Now, whenever you try to add a new resource to this API, CentraSite performs an internal validation of the existing resource URI. Depending on the validation, CentraSite handles the HTTP request in the following way:

- **Endpoint exactly ends with the existing resource URI** - Resource URI `/phones`. In this case, the URI is automatically substituted with the tokenizer. This ensures that the Mediator processes the HTTP request and routes the request to the appropriate native API endpoint for the requested resource.
- **Endpoint does not exactly end with the existing resource URI** - Resource URI `/invoke`. In this case, the URI is not substituted with the tokenizer and eventually results in a failure alert.

Now, based on the routing configuration for the native API, you have the following workaround options:

Straight-Through Routing Action

Given the straight-through routing action, you have two options:

Workaround Option 1:

Modify the Straight-Through routing action of the native REST API.

To modify the Straight-Through routing action

1. In CentraSite Business UI, display the details page for the native REST API. If you need procedures for this step, see *Working with REST-based APIs in CentraSite*.
2. In the action bar for the API, click **Virtualize**.
3. In the **Virtualize <API_Name> (Step 1 of 3)** wizard, select the virtual alias and its endpoint for reconfiguration.
4. Click **Next**.
5. In the **Virtualize <API_Name> (Step 2 of 3)** wizard, locate the “Straight-Through” routing action in the Message Flow section.
6. Mouse hover the action name, and click the **Configure** icon to the right of the name. This opens the **Straight-Through Routing** dialog box.
7. In the **Route to** endpoint field, modify the existing endpoint. Append the resource path tokenizer `${sys:resource_path}` with the base URL in the format `<base-url>/${sys:resource_path}`.

Thus, for our sample endpoint:

```
http://www.phonestore.com/api/phones
```

The modified endpoint should be:

```
http://www.phonestore.com/api/${sys:resource_path}
```

8. Click **OK** and save the modified API.
9. If you choose to use this option, in addition to the above steps in CentraSite Business UI, you must consider the following:
 - Modify the native service implementation to reconfigure the endpoint and resource URI.
 - -OR-
 - In the details page of the native REST API, edit the resource URI to exactly match with the endpoint. Thus for our example, modify the resource URI to read `/phones`. In addition, you must inform the clients of changes that have been made to the endpoint and resource URI.

Workaround Option 2:

Convert the Straight-Through Routing action of the native REST API to either a Context-Based Routing or Content-Based Routing action.

To convert the Straight-Through routing action

1. In CentraSite Business UI, display the details page for the native REST API. If you need procedures for this step, see *Working with REST-based APIs in CentraSite*.
2. In the action bar for the API, click **Virtualize**.
3. In the **Virtualize <API_Name> (Step 1 of 3)** wizard, select the virtual alias and its endpoint for reconfiguration.
4. Click **Next**.
5. In the **Virtualize <API_Name> (Step 2 of 3)** wizard, locate the “Straight-Through” routing action in the Message Flow section.
6. Mouse hover the action name, and click the **Delete** icon to the right of the name.
7. Drag and drop the Content-Based Routing action or the Context-Based Routing action from the Policy Actions accordion to the Message Flow section.
8. Configure the Content-Based Routing action or the Context-Based Routing action as described in the subsequent sections.

Content-Based Routing Action

Given the Content-Based Routing action, specify a custom routing rule. Perform the following steps:

1. In CentraSite Business UI, display the details page for the native REST API. If you need procedures for this step, see *Working with REST-based APIs in CentraSite*.
2. In the action bar for the API, click **Virtualize**.
3. In the **Virtualize <API_Name> (Step 1 of 3)** wizard, select the virtual alias and its endpoint for reconfiguration.
4. Click **Next**.
5. In the **Virtualize <API_Name> (Step 2 of 3)** wizard, locate the “Content-Based” routing action in the Message Flow section.
6. Mouse hover the action name, and click the **Configure** icon to the right of the name. This opens the **Content-Based Routing** dialog box.
7. In the **Default Route to** endpoint field, modify the existing endpoint. Append the resource path tokenizer `${sys:resource_path}` with the base URL in the format `<base-url>/${sys:resource_path}`.

Thus, for our sample endpoint:

```
http://www.phonestore.com/api/phones
```

The modified endpoint should be:

```
http://www.phonestore.com/api/${sys:resource_path}
```

8. Click the **Add Routing Rule** button. In the **Routing Rule** dialog box, complete the following fields:
 - a. In the **XPath Expression** field, specify an XPath expression unique to the resource.
 - b. Add a namespace value in the **Prefix** and **URI** fields.
 - c. In the **Route To** field, specify the native API endpoint.

In our sample,

```
http://www.phonestore.com/api/phones
```

- d. Click **OK** to save the new routing rule.
9. Click **OK** and save the modified API.

Context-Based Routing Action

Given the Context-Based Routing action, specify a custom routing rule. Perform the following steps:

1. In CentraSite Business UI, display the details page for the native REST API. If you need procedures for this step, see *Working with REST-based APIs in CentraSite*.
2. In the action bar for the API, click **Virtualize**.
3. In the **Virtualize <API_Name> (Step 1 of 3)** wizard, select the virtual alias and its endpoint for reconfiguration.
4. Click **Next**.
5. In the **Virtualize <API_Name> (Step 2 of 3)** wizard, locate the “Context-Based” routing action in the Message Flow section.
6. Mouse hover the action name, and click the **Configure** icon to the right of the name. This opens the **Context-Based Routing** dialog box.
7. In the **Default Route to** endpoint field, modify the existing endpoint. Append the resource path tokenizer `${sys:resource_path}` with the base URL in the format `<base-url>/${sys:resource_path}`.

Thus, for our sample endpoint:

```
http://www.phonestore.com/api/phones
```

The modified endpoint should be:

```
http://www.phonestore.com/api/${sys:resource_path}
```

8. Click the **Add Routing Rule** button.
 - a. Enter a name for the new routing rule.
 - b. For the **Variable** field, select **Predefined Context Variable** from the drop-down list.

- c. In the **Condition** section of the dialog box, take the following steps.
 - a. Select `String` in the first **Data Type** field.
 - b. Select the `Virtual Service Operation` in the second **Predefined Context** field.
 - c. Specify the `Equal To` operator in the third field.
 - d. Specify the exact resource name in the **Variable Value** field. In our example, `invoke`.
 - e. In the **Route To** field, specify the native API endpoint.
In our sample,
`http://www.phonestore.com/api/phones`
 - f. Click **OK** to save the new routing rule.
9. Click **OK** and save the modified API.

11 Run-Time Governance Reference

■ Run-Time Events and Key Performance Indicator (KPI) Metrics	328
■ Built-In Run-Time Actions Reference for Virtual Services	347
■ Built-In Run-Time Actions Reference for APIs	382
■ Computed Runtime Actions	472

Run-Time Events and Key Performance Indicator (KPI) Metrics

CentraSite can receive run-time events and Key Performance Indicator (KPI) metrics. A run-time event is an event that occurs while services are actively deployed on the gateway. Examples of run-time events include:

- Successful or unsuccessful SOAP requests/responses.
- Policy violation events, which are generated upon violation of service's run-time policy.
- Service monitoring events, which are generated by the service-monitoring actions in the run-time policy.

KPI metrics are used to monitor the run-time execution of virtual services. Metrics include the maximum response time, average response time, fault count, availability of virtual services, and more. If you include run-time monitoring actions in your run-time policies, the actions will monitor the KPI metrics for virtual services, and can send alerts to various destinations when user-specified performance conditions for a service are violated.

CentraSite provides predefined event types for use with any supported policy-enforcement point (PEP), such as webMethods Mediator. In addition, you can create custom event types.

The run-time event data are collected by the PEP and published to CentraSite via SNMP. The PEP publishes data for all run-time events for all instances of the PEP gateway.

You can view the run-time events and metrics on the CentraSite Control user interface. You can view them for all targets, for a particular target, or for a particular virtual service.

The Run-Time Event Types

The types of run-time events that Mediator can publish are as follows:

Event Type	Description
Lifecycle	A Lifecycle event occurs each time Mediator is started or shut down.
Error	An Error event occurs each time an invocation of a virtual service results in an error.

Event Type	Description
Policy Violation	A Policy Violation event occurs each time an invocation of a virtual service violates a run-time policy that was set for the virtual service.
Transaction	A Transaction event occurs each time a virtual service is invoked (successfully or unsuccessfully).
Monitoring	Mediator publishes key performance indicator (KPI) metrics, such as the average response time, fault count, and availability of all virtual services (described below).

The Key Performance Indicator (KPI) Metrics

For the Monitoring event type, Mediator can publish the following types of KPI metrics:

Metric	Reports on...
Availability	The percentage of time that a virtual service was available during the current interval. A value of 100 indicates that the service was always available. Only the time when the service is unavailable counts against this metric. If invocations fail due to policy violations, this parameter could still be as high as 100.
Average Response Time	The average amount of time it took the service to complete all invocations in the current interval. This is measured from the moment Mediator receives the request until the moment it returns the response to the caller.
Fault Count	The number of failed invocations in the current interval.
Maximum Response Time	The maximum amount of time it took the service to complete an invocation in the current interval.
Minimum Response Time	The minimum amount of time it took the service to complete an invocation in the current interval.
Successful Request Count	The number of successful service invocations in the current interval.

Metric	Reports on...
Total Request Count	The total number of requests for each service running in Mediator in the current interval.
Note:	By default, Average Response Time, Minimum Response Time, and Maximum Response Time do not include metrics for failed invocations. You can include metrics for failed invocations by setting the <code>pg.PgMetricsFormatter.includeFaults</code> parameter to true. For more information about advanced settings, see <i>Administering webMethods Mediator</i> .

The Event Notification Destinations

Mediator can publish data about the run-time events and metrics to the following destinations:

- An SNMP server. You can use one or both of the following kinds of servers:
 - CentraSite's SNMP server, which uses SNMPv3 user-security model.
For the procedure to configure Mediator to send SNMP traps to the CentraSite SNMP server, see *Administering webMethods Mediator*.
 - A third-party SNMP server, which uses either the SNMPv1 community-based security model or the SNMPv3 user-based security model.
For the procedure to configure Mediator to send SNMP traps to a third-party SNMP server, see *Administering webMethods Mediator*.
- An EDA destination. Mediator can use EDA to publish run-time events and metrics to a database. Mediator uses a JDBC connection pool that you need to define in the Integration Server.
For the procedure to configure Mediator to send this data to an EDA destination, see *Administering webMethods Mediator*.

Destinations for the Monitoring and Transaction Events

For the Monitoring and Transaction event types, there are additional event notification destinations to choose from (in addition to the EDA and SNMP destinations).

Monitoring events are generated by the following run-time actions that you can configure for your virtual services in CentraSite:

- Monitor Service Performance.
- Monitor Service Level Agreement.
- Throttling Traffic Optimization.

Transaction events are generated by the run-time action Log Invocations.

The available destinations for Monitoring and Transaction events are:

- An EDA destination (a database).
- The CentraSite SNMP server or a third-party SNMP server.
- The virtual service's Events profile in CentraSite.
- An SMTP email server.
- Your Integration Server's local log.
- Your Integration Server's audit log (for Transaction events only).

You will select these destinations when you configure your virtual services in CentraSite.

These additional destinations for the monitoring and transaction events are described below.

- [SMTP Email Servers](#)
- [The Integration Server's Local Log](#)
- [The Integration Server's Audit Log](#)

SMTP Email Servers

To specify an SMTP email destination, you must:

- Select the “Email” option as a destination when you configure the run-time actions listed above.
- Set the “Email Configuration” parameters in Integration Server Administrator (go to **Solutions > Mediator > Administration > Email**) as described in *Administering webMethods Mediator*.

The Integration Server's Local Log

To specify the Integration Server's local log as a destination, you must:

- Select the “Local Log” option as a destination when you configure the run-time actions listed above. When configuring the actions, you must also specify the severity of the messages to be logged (the logging level).
- Set the Integration Server Administrator's logging level for Mediator to match the logging levels specified for the run-time actions (go to **Settings > Logging > Server Logger**). For example, if a “Log Invocation” action is set to the logging level of Error, you must also set Integration Server Administrator's logging level for Mediator to Error. If the action's logging level is set to a low level (Warning-level or Information-level), but Integration Server Administrator's logging level for Mediator is set to a higher level (Error-level), then only the higher-level messages are written to the log file.

Entries posted to the local log are identified by a product code of MED.

The Integration Server's Audit Log

You can select the Integration Server Audit Log as a destination for the “Log Invocation” action only. If you expect a high volume of invocations in your system, it is recommended that you select the Audit Log destination. For more information, see *webMethods Audit Logging Guide*.

The Metrics Tracking Interval

Mediator tracks performance metrics by intervals. The interval is a period of time you set in Mediator, during which metrics are collected for reporting to CentraSite. You set the interval in the Publish Interval field on the **Mediator > Administration > CentraSite Communication** page in the Integration Server Administrator. For details, see *Administering webMethods Mediator*.

Mediator only tracks metrics for the current interval. At the end of the interval, Mediator aggregates the metrics and reports them to CentraSite. Once the metrics are reported, Mediator resets its counters for the new interval. Mediator does not calculate and aggregate metrics across intervals. If Mediator is shut down or the virtual service is undeployed before the current interval expires, the performance data is discarded.

Note: To avoid the need for Mediator to store metrics during periods of inactivity, Mediator stores only first and last zero value metrics that occurs during an interval, and discards the remaining consecutive zero value metrics. Doing this drastically reduces the storage space consumed by the metrics, and speeds the queries you perform in the dashboard. Skipping the in-between zero metrics will not affect in the performance graphs shown in the dashboard.

For more information about the metrics tracking interval, see *Administering webMethods Mediator*.

Configuring CentraSite to Receive Run-Time Events and Metrics

Prerequisites:

- Ensure that Mediator is configured for publishing events to an SNMP server, as described in *Administering webMethods Mediator*.
- If you use a target type other than Mediator or webMethods Insight, be sure to configure CentraSite to publish events by providing your own MIB file in your target type's definition file, as described in ["Run-Time Gateways" on page 59](#). (CentraSite provides a MIB file for Mediator and Insight.)
- Optionally change CentraSite's default settings for logging run-time events, as described in the *CentraSite Administrator's Guide*. By default, CentraSite logs all predefined event types, but you may disable any type.

CentraSite provides an Event Receiver, which is a data collector that collects the run-time event data. The Event Receiver listens for run-time events from the target instances via the SNMP (Application-Layer) protocol, and contains the logic to parse and store event data in the Event Receiver's data store.

The following sections describe how to configure the Event Receiver's properties file:

- [Components of the Event Receiver](#)
- [Configuring the Event Receiver](#)
- [Event Type Modeling](#)
- [Event Modeling](#)

Components of the Event Receiver

The Event Receiver contains the following components.

■ The SNMP Listener

CentraSite's SNMPv3 Trap Listener, which supports SNMP4J. This Listener starts automatically when CentraSite starts.

■ The Intermediate Queue

The queue from the SNMP Listener to the Event Processor. This queue decouples the SNMP Listener threads from the Event Processor to improve throughput. The following modes are supported.

- **FileSystem:** Incoming Traps will be stored temporarily in the file system
- **InMemory:** Incoming Traps will be stored temporarily in memory
- **NoQueue:** Incoming Traps will not be stored in any intermediate queue; the SNMP Listener threads will be processed.

To select the mode, set the `eventsQueueImpl` property as described in "[Setting the Events Queue Implementation Property](#)" on page 337.

■ The Event Processor

The Event Processor (`SOALinkSNMPEventsListener`) transforms incoming SNMPv3 Traps into an XML file (`Events.xml`) that complies with the schema in the `RuntimeEvents` Collection component. The Event Processor transforms an SNMPv3 Trap to the `Events.xml` file as follows:

1. Determines the Event Type (and Target Type) to which the Trap belongs, and gets the corresponding UUIDs. This involves searching all Event Type-to-Trap mappings in all the defined target types, using the Trap's OID. Since this is an expensive search, the Event Type-to-Trap mapping is cached to improve performance.
2. Parses the Trap attributes and obtains: the Service (UUID), the Target (Name), the TimeStamp, and the SessionId. The Processor then searches the registry/

repository and obtains the corresponding UUID for the Target Name. This mapping is also cached to improve performance.

3. Collects the remaining attributes from the Trap.
4. Constructs the Events.xml file using the Event Type UUID, Target Type UUID, Service UUID, Target UUID, TimeStamp, SessionId, and other collected attributes.

■ **The Batch Condition**

The Batch Condition is a set of OR conditions used by the Event Processor. The Event Processor supports two modes of event storage into CentraSite: BatchMode and NoBatchMode. BatchMode is available only for FileSystem and InMemory queues. When BatchMode is enabled, the Event Processor continues to accumulate Events.xml documents until one of the conditions is evaluated as true. Then it inserts all the documents as a single batch into CentraSite.

To specify BatchMode or NoBatchMode, set the batch-related properties as described in ["Setting the Properties for FileSystem or InMemory" on page 338](#).

■ **The RuntimeEvents Collection**

The run-time events are stored in the RuntimeEvents Collection as non-registry objects. For information about how events are stored, see ["Event Type Modeling" on page 339](#).

Configuring the Event Receiver

The Event Receiver is bundled in the installation as a Web-Application named SOALinkSNMPEventsListener supporting the JavaEE standard. The configuration file for the Event Receiver is located here:

```
<CentraSiteInstallDir> /cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml
```

The web.xml configuration file contains all the Event Receiver configuration properties. You must set these properties as described below, and then restart CentraSite.

- [Setting the Database Configuration Properties](#)
- [Setting the SNMPv3 Transport Configuration Properties](#)
- [Setting the SNMPv3 USM Configuration Properties](#)
- [Setting the Events Queue Implementation Property](#)
- [Setting the Properties for FileSystem or InMemory](#)

Setting the Database Configuration Properties

In the Event Receiver's configuration file, set the following properties related to the RuntimeEvents Collection database.

Database Property and Description

com.softwareag.centrasite.soalink.events.dbUrl

The URL of the RuntimeEvents Collection database. All run-time events will be persisted to this database.

com.softwareag.centrasite.soalink.events.dbUserId

The user name that the Events Listener will use for authentication before persisting event data to the RuntimeEvents Collection database. The default value of this property is the predefined user EventsUser.

Optionally, you can change the value EventsUser to any login user who has the following privileges:

- Write access on the Tamino collection "RuntimeEvents".
- Read access on "TargetTypes", "Targets", "RuntimeEventTypes", and "LogUnit", which are under the Tamino collection "CentraSite".

If you want to change the value to a login user, enter that login user's name in the form `<hostName>\<userName>`.

Important: The predefined password of EventsUser is EventsManager4CS (there is no need to specify the password in this file). If you want to change this password, or if you have changed the value EventsUser to a login user, *you must change the password*. For details about users, groups, roles, and permissions, see *Getting Started with CentraSite*. Whenever you change the password, you must restart CentraSite.

com.softwareag.centrasite.soalink.events.dbNonActivityTimeOut

The non-activity timeout in seconds for the RuntimeEvents Collection database (default 2592000 seconds (that is, 30 days)).

Setting the SNMPv3 Transport Configuration Properties

In the Event Receiver's configuration file, set the following properties related to a SNMPv3 Transport.

SNMPv3 Transport Property and Description

com.softwareag.centrasite.soalink.events.snmp.transport

Wire transport protocol that will be used by the SNMP Listener. Supported values are: TCP and UDP.

com.softwareag.centrasite.soalink.events.snmp.host

SNMPv3 Transport Property and Description

The CentraSite host name or IP address to which the SNMP listener will bind.

com.softwareag.centrasite.soalink.events.snmp.port

The port to which the SNMP listener will bind. The default is 8181. If Microsoft Internet Information Services (IIS) is installed (or will be installed) on the same machine hosting Integration Server/Mediator, then you may want to change the default SNMP port of 8181 to something else, to avoid any potential runtime conflicts when sending SNMP packets.

com.softwareag.centrasite.soalink.events.snmp.maxInboundMessageSizeInBytes

Maximum inbound message size in bytes (an integer). Traps that exceed this size limit will be rejected. Default value is 256Kb.

com.softwareag.centrasite.soalink.events.snmp.dispatcherPoolSize

The SNMP Listener's Worker-Thread pool size (default is 10). This determines the throughput of the Listener.

Setting the SNMPv3 USM Configuration Properties

In the Event Receiver's configuration file, set the following properties related to SNMPv3 USM.

SNMPv3 USM Property and Description

com.softwareag.centrasite.soalink.events.snmp.engineId

EngineId to be used by the SNMP Listener. If the parameter is left blank, the SNMP Listener will auto-generate the engineId.

com.softwareag.centrasite.soalink.events.snmp.securityName

The SecurityName to be used by the SNMP Listener.

com.softwareag.centrasite.soalink.events.snmp.securityLevel

The Maximum SecurityLevel to be supported by SNMP Listener. Supported values in order are: NOAUTH_NOPRIV, AUTH_NOPRIV, and AUTH_PRIV. For example, AUTH_PRIV provides the highest level of security but also supports the other two levels. Similarly AUTH_NOPRIV supports NOAUTH_NOPRIV.

com.softwareag.centrasite.soalink.events.snmp.authProtocol

SNMPv3 USM Property and Description

AuthorizationProtocol to be used by the SNMP Listener for decoding the incoming trap. Supported values are: MD5 and SHA.

com.softwareag.centrasite.soalink.events.snmp.authPassPhraseKey

The PassPhrase key to be used by the AuthorizationProtocol. The passphrase key length should be ≥ 8 . The key is stored in this file; the passphrase value is stored securely in passman.

Note: The value should be the same as the authPassphrasKey that you have set in the `web.xml` file. You can also use the `CentraSiteCommand.cmd` tool to reset this key at a later stage as required.

com.softwareag.centrasite.soalink.events.snmp.privProtocol

The PrivacyProtocol to be used by the SNMP Listener for decoding the incoming trap. Supported values are: DES, AES128, AES, AES192, AES256, 3DES, and DESEDE.

com.softwareag.centrasite.soalink.events.snmp.privPassPhraseKey

The PassPhrase key to be used by the PrivacyProtocol. The passphrase length should be ≥ 8 . The key is stored in this file; the passphrase value is stored securely in passman.

Note: The value should be the same as the privPassphrasKey that you have set in the `web.xml` file. You can also use the `CentraSiteCommand.cmd` tool to reset this key at a later stage as required.

Setting the Events Queue Implementation Property

In the Event Receiver's configuration file, set the following property related to the implementation of the events queue.

Events Queue Property and Description

com.softwareag.centrasite.soalink.events.eventsQueueImpl

Supported values are:

- **FileSystem:** Incoming Traps will be stored temporarily in the file system
- **InMemory:** Incoming Traps will be stored temporarily in memory
- **NoQueue:** Incoming Traps will not be stored in any intermediate queue; the SNMP Listener threads will be processed one by one

Events Queue Property and Description

Additional, related properties are described in ["Setting the Properties for FileSystem or InMemory"](#) on page 338.

Setting the Properties for FileSystem or InMemory

When the `eventsQueueImpl` property is set to either `FileSystem` or `InMemory`, you should also set the following properties.

Property for FileSystem or InMemory and Description

`com.softwareag.centrasite.soalink.events.enableBatchInsertion`

Enable or disable batch insertion of events into the database. Supported values are `true` and `false`. If `true`, events will be batched as per the "batching rules" properties below, and the batch will be stored to the database. If `false`, events will be stored to the database one by one.

`com.softwareag.centrasite.soalink.events.maxNumOfEventsPerBatch`

Maximum number of events in a batch. Should be an integer value. A value ≤ 0 disables this rule. This rule is evaluated only on arrival of a new Trap.

`com.softwareag.centrasite.soalink.events.maxSizeOfBatch`

Maximum size (in bytes) of a batch. Default value is 512KB. Should be an integer value. A value ≤ 0 disables this rule. This rule is evaluated only on arrival of a new Trap.

`com.softwareag.centrasite.soalink.events.maxTimeIntervalBetweenBatches`

Maximum time interval (in milliseconds) between two subsequent batch storages. Should be an integer value. A value ≤ 0 disables this rule. Unlike the other two rules, this rule is evaluated periodically. Hence this rule prevents any trap stuck in the batch for ever if inflow of traps stops; in short this acts as a batch-timeout. A very low value for this rule reduces batch efficiency and introduces unnecessary looping.

`com.softwareag.centrasite.soalink.events.fileSystemQueueDir`

(Only applies when the `eventsQueueImpl` property is set to `FileSystem`.) The directory that should be used as `FileSystem Queue`. Incoming traps will be stored in this directory temporarily and hence should have write permission. The path can be absolute or relative. It is advisable to provide the absolute path. Relative paths will be considered relative to one of the following, based on availability in the same order:

Property for FileSystem or InMemory and Description

1. SOALinkSNMPEventsListener/WEB-INF directory for exploded deployments.
2. javax.servlet.context.tempdir for zipped deployments.
3. java.io.tmpdir if none of the above are available.

Event Type Modeling

Event types are modeled as registry objects. The String, Date, Integer and Boolean event attributes are stored in the registry/repository as slots. The File-Type attributes (representing payloads/binary-data) are stored as HasExternalLink associations.

For example, consider the predefined event type Transaction. If you go to the **Target Type** details page, you will see the Transaction event type attributes (which are obtained from the webMethodsESB.mib file) as follows:

Attribute Name	Object ID	Type
Service	1.3.6.1.4.1.1783.201.1.1.1	String
Target	1.3.6.1.4.1.1783.201.1.1.2	String
Timestamp	1.3.6.1.4.1.1783.201.1.1.3	Date
Consumer	1.3.6.1.4.1.1783.201.1.1.4	String
RequestStatus	1.3.6.1.4.1.1783.201.1.1.5	String
ResponsePayload	1.3.6.1.4.1.1783.201.1.1.6	File
RequestPayload	1.3.6.1.4.1.1783.201.1.1.7	File
ProviderRoundTripTime	1.3.6.1.4.1.1783.201.1.1.8	Integer
TotalRoundTripTime	1.3.6.1.4.1.1783.201.1.1.9	Integer
SessionID	1.3.6.1.4.1.1783.201.1.1.16	String
ConsumerIP	1.3.6.1.4.1.1783.201.1.1.17	String
OperationName	1.3.6.1.4.1.1783.201.1.1.21	String

Attribute Name	Object ID	Type
NativeEndpoint	1.3.6.1.4.1.1783.201.1.1.22	String

All of these attributes (except the File-Type attributes RequestPayload and ResponsePayload) are stored as registry object slots, as follows:

Slot Key	Slot Type	Slot Value (Attribute)
uddi_16d34470-9a92-11dd-9b43-e319c2a6593c	xs:string	Service
uddi_f18b5a40-9a91-11dd-b95e-b4758b17b88b	xs:string	Target
uddi_c798d3c0-9a91-11dd-889e-b999c87ba6b7	xs:datetime	TimeStamp
uddi_a7476ff0-a108-11dd-9c38-d8fd010529cc	xs:string	Consumer
uddi_a7476ff0-a108-11dd-9c38-eac6d60fc855	xs:string	RequestStatus
uddi_a7476ff0-a108-11dd-9c38-f3f84c6111f0	xs:integer	ProviderRoundTrip Time
uddi_a7476ff0-a108-11dd-9c38-d02170b3aae3	xs:integer	TotalRoundTripTime
uddi_21b67010-9a92-11dd-926a-991c4c180c79	xs:string	SessionID
uddi_a7476ff0-a108-11dd-9c38-d34f346cb3d5	xs:string	ConsumerIP
uddi_f1c8a185-4b18-4974-a360-6c70756a174a	xs:string	OperationName
uddi_524d05f5-d526-4605-b594-ace1cb750d33	xs:string	NativeEndpoint

The File-Type attributes ResponsePayload and RequestPayload are stored as HasExternalLink associations, as follows:

Association Key	Association Name (Attribute)
uddi:a747704b-a108-11dd-9c38-fde9d932116a	ResponsePayload

Association Key	Association Name (Attribute)
uddi:a745265b-a108-11dd-9c38-bf43eee17363	RequestPayload

The “Target Type to Event Type Association” Object

A target type (represented as a concept) is associated with an event type (represented as a registry object) by a “Target Type to Event Type Association” object, which defines the “UUID to MIB OID” mapping.

The following table shows the contents of a sample object that associates the target type webMethods Mediator with the event type Transaction. The table's columns are described below.

- **Attribute:** The Attribute column is not part of the object; it is included here simply for your reference.
- **Slot Key:** Contains the UUID, which is obtained from the event type registry object.
- **Slot Type:** Contains the slot type, which is obtained from the event type registry object.
- **Slot Value:** Contains the event type attribute's Object Identifier (OID), which is obtained from the MIB file.

Attribute	Slot Key (Event Type UUID)	Slot Type	Slot Value (Event Attribute OID)
Service	uddi_16d34470-9a92-11dd-9b43-e319c2a6593c	xs:string	1.3.6.1.4.1.1783.201.1.1.1
Target	uddi_f18b5a40-9a91-11dd-b95e-b4758b17b88b	xs:string	1.3.6.1.4.1.1783.201.1.1.2
TimeStamp	uddi_c798d3c0-9a91-11dd-889e-b999c87ba6b7	xs:datetime	1.3.6.1.4.1.1783.201.1.1.3
Consumer	uddi_a7476ff0-a108-11dd-9c38-d8fd010529cc	xs:string	1.3.6.1.4.1.1783.201.1.1.4
RequestStatus	uddi_a7476ff0-a108-11dd-9c38-eac6d60fc855	xs:string	1.3.6.1.4.1.1783.201.1.1.5

Attribute	Slot Key (Event Type UUID)	Slot Type	Slot Value (Event Attribute OID)
ResponsePayload	uddi_a747704b-a108-11dd-9c38-fde9d932116a	xs:anyURI	1.3.6.1.4.1.1783.201.1.1.6
RequestPayload	uddi_a745265b-a108-11dd-9c38-bf43eee17363	xs:anyURI	1.3.6.1.4.1.1783.201.1.1.7
ProviderRound TripTime	uddi_a7476ff0-a108-11dd-9c38-f3f84c6111f0	xs:integer	1.3.6.1.4.1.1783.201.1.1.8
TotalRoundTrip Time	uddi_a7476ff0-a108-11dd-9c38-d02170b3aae3	xs:integer	1.3.6.1.4.1.1783.201.1.1.9
SessionID	uddi_21b67010-9a92-11dd-926a-991c4c180c79	xs:string	1.3.6.1.4.1.1783.201.1.1.16
ConsumerIP	uddi_a7476ff0-a108-11dd-9c38-d34f346cb3d5	xs:string	1.3.6.1.4.1.1783.201.1.1.17
OperationName	uddi_f1c8a185-4b18-4974-a360-6c70756a174a	xs:string	1.3.6.1.4.1.1783.201.1.1.21
NativeEndpoint	uddi_524d05f5-d526-4605-b594-ace1cb750d33	xs:string	1.3.6.1.4.1.1783.201.1.1.22

Event Modeling

An event is an instance of an event type. Events are modeled in a separate schema from the event type schema. CentraSite models events as non-registry objects (to avoid storing large amounts of unwanted event data in the registry/repository), and instead stores event data in a database collection within the Event Receiver. CentraSite maps events to their corresponding event types, using the event types' UUIDs. Similarly, events are mapped to target types, targets and services using UUIDs and the event type attributes.

The stored event data will contain:

- The event Trap ID (MIB OID).
- The event Trap value, which consists of:
 - The attribute key (MIB OID).
 - The attribute value.

The event data is stored in the Event Receiver as an "events" doctype.

If an event contains payloads (for example, File-Type attributes such as ResponsePayload and RequestPayload), the payloads are stored in the Event Receiver as a "payloads" doctype, and will be referenced by the event stored under the "event" doctype, using ino:id. This is used to reduce de-serialization of the usually large payloads, and to improve performance of queries on the stored events.

Viewing Run-Time Events and Metrics

You can view the run-time events and metrics that occurred for:

- A particular target or all targets (see ["Viewing Run-Time Events and Metrics for Targets" on page 343](#)).
- Each virtual service (see ["Viewing Run-Time Events and Metrics for Virtual Services" on page 345](#)).
- Each API (see ["Viewing Run-Time Events and Metrics for APIs" on page 345](#)).

Viewing Run-Time Events and Metrics for Targets

Use the following procedure to view lists of run-time events for a particular target or for all targets.

If you are using the Mediator target, ensure that Mediator is configured to send event notifications to the destination(s) that are applicable for each event type. For details, see *Administering webMethods Mediator*.

Note: You must have the permissions to manage targets, as described in ["Run-Time Gateways" on page 59](#).

To view a list of run-time events for targets

1. In CentraSite Control, go to **Operations > Events > Event List**.
2. Use the following fields to filter the event list you want to view:

In this field...	Specify...
Target Type	The type of the target whose events you want to view.

In this field...	Specify...
Target	The target whose events you want to view (or select All to view events of all targets).
Event Type	A particular event type, or select All to view all event types. For descriptions of the predefined event types, see " The Run-Time Event Types " on page 328.
Service Type	Select All or Virtual Service . Note: CentraSite does not provide out-of-the-box policy-enforcement for web services.
Date Range	A range of dates from which to view the events.
Start Date	Alternatively, select the check box next to this field and click the calendar and select a starting date and time.
End Date	Click the calendar and select an ending date and time.

- Click the **Search** button.
- The generated event list displays the following information:

Field	Description
Date/Time	The date/time that the event occurred. Click this hyperlinked value to view the Event Detail page, which will contain the event's SOAP request or response name in the Attribute column. Click the hyperlinked request or response name to display the full SOAP request or response.
Session ID	<i>Read-only.</i> The session ID that generated the event.
Event Type	<i>Read-only.</i> The type of event (for example, Monitoring, Policy Violation, Error, and so on).
Service Name	<i>Read-only.</i> The name of the service that caused the event.
Service Type	<i>Read-only.</i> The service's type.
Target	<i>Read-only.</i> The target on which the event occurred.
Target Type	<i>Read-only.</i> The type of the target on which the event occurred.

Note: To view the list of attributes that are mapped for each event type, go to the target type's detail page (see ["Run-Time Gateways" on page 59](#)).

Viewing Run-Time Events and Metrics for Virtual Services

You can view the events and metrics for a virtual service in its Events profile and its Performance profile. For details, see ["Virtualized Services in CentraSite " on page 67](#).

Viewing Run-Time Events and Metrics for APIs

You can view the events and metrics for an API in its Runtime Events profile and its Runtime Metrics profile. For details, see ["Virtualized Services in CentraSite " on page 67](#).

Creating Custom Run-Time Events

CentraSite provides the predefined event types described in ["The Run-Time Event Types" on page 328](#). In addition, you can create custom run-time events that CentraSite will monitor.

Prerequisite: You must have the Manage Runtime Event Types permission. By default, the predefined roles CentraSite Administrator and Operations Administrator include this permission. For more information about roles and permissions, see *Getting Started with CentraSite*.

Important: To enable CentraSite to recognize custom event types, ensure that your MIB file (which is contained in your target type definition file) contains the SNMP Traps metadata and Object Identifiers for the custom events. For more information, see ["Run-Time Gateways" on page 59](#).

To create custom event types

1. In CentraSite Control, go to **Operations > Events > Event Types** to display the **Event Types** page.

The page displays all the predefined event types (Monitoring, Policy Violation, Transaction, Error, and Lifecycle) and any custom event types that have been defined.

2. To view the details of any event type, click its hyperlinked name.

The list of attributes for the event type is displayed. You can edit the attributes of custom event types, but not the predefined event types (see ["Modifying Run-Time Events" on page 346](#)).

3. To create a custom event type, click the **Add Event Type** button. In the **Add/Edit Event Type** page specify a name and description for the event type. Event type names can contain any character (including spaces), and are not case-sensitive.

4. In the Event Type Attribute panel, the following default attributes are displayed. These attributes are required and cannot be deleted.

Attribute	Data Type
TimeStamp	Date
Target	String
Service	String
SessionID	String

To create additional attributes, perform the following steps:

- a. Click the plus button at the bottom of the attribute list.
- b. Specify a name in the **Name** column and a value in the **Data Type** column (Boolean, File, Date, Integer, or String). Attribute names can contain any character (including spaces).
- c. To add another attribute, click the plus button at the bottom of the list.
- d. To delete an attribute, click the minus button for the attribute you want to delete.
- e. Click **Save**.

Modifying Run-Time Events

To edit and delete custom event types, perform the following steps.

To modify a custom run-time event

1. In CentraSite Control, go to **Operations > Events > Event Types** to display the **Event Types** page.
The page displays all event types that have been defined.
2. To delete a custom event type, select the check box next to the event type and click the **Delete** button.
3. To edit the attributes of a custom event type, perform the following steps:
 - a. Click its hyperlinked name to display the **Add/Edit Event Type** page.
 - b. You can change the value of an attribute's data type, but not its name. Data types can be Boolean, File, Date, Integer, or String.
 - c. To add another attribute, use the plus button at the bottom of the list.
 - d. To delete an attribute, click the minus button next to the attribute.
 - e. Click **Save**.

Built-In Run-Time Actions Reference for Virtual Services

This section describes the built-in run-time actions that you can include in run-time policies for virtual services. You use these actions only when you are using CentraSite Control to create run-time policies for virtual services. The content is organized under the following sections:

Summary of the Run-Time Actions for Virtual Services

You can include the following kinds of built-in run-time actions in the run-time policies for virtual services:

- [WS-SecurityPolicy 1.2 Actions](#)
- [Monitoring Actions](#)
- [Additional Actions](#)

WS-SecurityPolicy 1.2 Actions

Mediator provides two kinds of actions that support WS-SecurityPolicy 1.2: authentication actions and XML security actions.

Authentication Actions (WS-SecurityPolicy 1.2)

Mediator uses the following authentication actions to verify that the requests for virtual services contain a specified WS-Security element:

Require WSS Username Token	Uses WS-SecurityPolicy authentication to validate user names and passwords that are transmitted in the SOAP message header for the WSS Username token.
Require WSS X.509 Token	Identifies consumers based on a WSS X.509 token.
Require WSS SAML Token	Uses a WSS Security Assertion Markup Language (SAML) assertion token to validate service consumers.

XML Security Actions (WS-SecurityPolicy 1.2)

These actions provide confidentiality (through encryption) and integrity (through signatures) for request and response messages.

Require Signing	Requires that a request's XML element (which is represented by an XPath expression) be signed.
------------------------	--

Require Encryption	Requires that a request's XML element (which is represented by an XPath expression) be encrypted.
Require SSL	Requires that requests be sent via SSL client certificates, and can be used by both SOAP and REST services.
Require Timestamps	Requires that timestamps be included in the request header. Mediator checks the timestamp value against the current time to ensure that the request is not an old message. This serves to protect your system against attempts at message tampering, such as replay attacks.

Monitoring Actions

Mediator provides the following run-time monitoring actions:

Monitor Service Performance	This action monitors a user-specified set of run-time performance conditions for a virtual service, and sends alerts to a specified destination when these performance conditions are violated.
Monitor Service Level Agreement	This action provides the same functionality as “Monitor Service Performance” but this action is different because it enables you to monitor a virtual service's run-time performance especially for particular consumer(s). You can configure this action to define a <i>Service Level Agreement (SLA)</i> , which is set of conditions that defines the level of performance that a specified consumer should expect from a service.
Throttling Traffic Optimization	(Not available in Mediator versions below 9.0.) This action limits the number of service invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated. You can use this action to avoid overloading the back-end services and their infrastructure, to limit specific consumers in terms of resource usage, and so on.

Additional Actions

Mediator provides the following actions, which you can use in conjunction with the actions above.

Identify Consumer	You use this action in conjunction with an authentication action (“Require WSS Username Token”, “Require WSS X.509 Token”, or “Require HTTP Basic Authentication”). Alternatively, you can
--------------------------	--

	use this action alone to identify consumers only by host name or IP address.
Require HTTP Basic Authentication	This action uses HTTP basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header against the Integration Server's user account.
Authorize User	This action authorizes consumers against a list of users and/or a list of groups registered in the Integration Server on which Mediator is running. You use this action in conjunction with an authentication action "Require WSS Username Token", "Require WSS SAML Token", or "Require HTTP Basic Authentication".
Authorize Against Registered Consumers	This action authorizes consumer applications against all consumer applications who are registered in CentraSite as consumers for the service.
Log Invocations	Logs request/response payloads to a destination you specify.
Validate Schema	Validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Configuring Destinations for Alerts and Logs

You can configure the destinations to which Mediator should send the alerts and transaction logging payloads generated by the built-in run-time actions that you include in run-time policies for virtual services.

The destinations can be configured at the:

- **Service level**, for the event types: Transaction Event and Monitoring Event
Refer to "Destinations for the Monitoring and Transaction Events" in *Run-Time Governance with CentraSite* for additional details on service level configuration.
- **Global level** (for all services), for metrics and the event types: Policy Violation Event, Error Event, and Lifecycle Event
Refer *Administering webMethods Mediator* for additional details on global level configuration.

You can configure one of the following destinations for alerts:

- CentraSite
- Local Log
- SNMP (CentraSite SNMP server or third-party SNMP server configured in Integration Server)

- E-mail
- EDA (to publish run-time events and KPI metrics to a database or a messaging server configured in Integration Server)

You can select one of the following destinations for transaction payloads:

- CentraSite
- Local Log
- SNMP (CentraSite SNMP server or third-party SNMP server configured in Integration Server)
- E-mail
- Audit Log (only for Log Invocation)
- EDA (to publish run-time events and KPI metrics to a database or a messaging server configured in Integration Server)

You can configure the EDA destination for the following policy actions:

- Log Invocation
- Monitor Service Level Agreement
- Monitor Service performance
- Throttling Traffic Optimization

The `watt.server.auth.skipForMediator` Property

This property specifies whether Integration Server authenticates requests for Mediator. You must set this property to true.

No request to Mediator should be authenticated by Integration Server. Instead, authentication should be handled by Mediator. Thus, to enable Mediator to authenticate requests, you must set `skipForMediator` to true (by default it is false).

When this parameter is set to true, Integration Server skips authentication for Mediator requests and allows the user credentials (of any type) to pass through so that Mediator can authenticate them. If you change the setting of this parameter, you must restart Integration Server for the changes to take effect.

To set `skipForMediator` to true

1. In the Integration Server Administrator, click **Settings > Extended**.
2. Click **Show and Hide Keys**.

Look for the `watt.server.auth.skipForMediator` property and ensure it is set to true.

3. If the `watt.server.auth.skipForMediator` property is not present, add it as follows:

- a. Click **Edit Extended Settings**.
- b. Type `watt.server.auth.skipForMediator=true` on a separate line.
- c. Click **Save**.
- d. Restart Integration Server.

Action Evaluation Order and Dependencies

When you deploy a virtual service, CentraSite automatically validates the service's run-time policy (or policies) to ensure that:

- Any action that appears in a single policy multiple times is allowed to appear multiple times.

For those actions that can appear in a policy only once (for example, Identify Consumer), Mediator will choose only one, which might cause problems or unintended results.

- All action dependencies are properly met. That is, some actions must be used in conjunction with another particular action.

CentraSite will inform you of any violation, and you will need to correct the violations before deploying the service.

Effective Policies

When you deploy a virtual service to Mediator, CentraSite combines the actions specified within the service's run-time policy (or policies) that apply to the virtual service, and generates what is called the *effective policy* for the virtual service. For example, suppose your virtual service is within the scope of two run-time policies: one policy that performs a logging action and another policy that performs a security action. When you deploy the virtual service, CentraSite automatically combines the two policies into one effective policy. The effective policy, which contains both the logging action and the security action, is the policy that CentraSite actually deploys to Mediator with the virtual service.

When CentraSite generates the effective policy, it validates the resulting action list to ensure that it contains no conflicting or incompatible actions. If the list contains conflicts or inconsistencies, CentraSite resolves them according to Policy Resolution Rules. For example, an action list can include only one Identify Consumer action. If the resulting action list contains multiple Identify Consumer actions, CentraSite resolves the conflict by including only one of the actions (selected according to a set of internal rules) in the effective policy and omitting the others.

The effective policy that CentraSite produces for a virtual service is contained in an object called a *virtual service definition* (VSD). The VSD is given to Mediator when you deploy the virtual service. After you deploy a virtual service, you can view its VSD (and thus examine the effective policy that CentraSite generated for it) from the CentraSite user interface or from the Mediator user interface.

The following table shows:

- The order in which Mediator evaluates the actions.
- Action dependencies (that is, whether an action must be used in conjunction with another particular action).
- Whether an action can be included multiple times in a single policy. If an action cannot be included multiple times in a single policy, Mediator selects just one for the effective policy, which may cause problems or unintended results.

Evaluation Order	Action	Dependency	Can include multiple times in a policy?
1	Require SSL	None.	If multiple actions appear, and one of them has its Client Certificate Required parameter set to Yes, only one occurrence of the action appears in the effective policy.
2	Require HTTP Basic Authentication	In Mediator versions below 9.0: None. In Mediator version 9.0 and above: Identify Consumer.	No. Mediator includes only one action in the effective policy.
3	Require WSS Username Token	Identify Consumer action.	No. Mediator includes only one action in the effective policy.
4	Require WSS X.509 Token	Identify Consumer action.	No. Mediator includes only one action in the effective policy.
5	Require WSS SAML Token	None.	No. Mediator includes only one action in the effective policy.
6	Require Signing	Identify Consumer action.	Yes. Mediator generates a UNION of all Require Signing actions for the effective policy.

Evaluation Order	Action	Dependency	Can include multiple times in a policy?
7	Require Encryption	Identify Consumer action.	Yes. Mediator generates a UNION of all Require Encryption actions for the effective policy.
8	Require Timestamps	Require Signing <i>and</i> Require Encryption.	No. Mediator includes only one action in the effective policy.
9	Identify Consumer	<p>If Identify Consumer's identifier field is set to:</p> <ul style="list-style-type: none"> ■ HTTP Authentication Token, the action Require HTTP Basic Authentication is also required. ■ WS-Security Authentication Token, the action Require WSS Username Token is also required. ■ Consumer Certificate, the actions Require WSS X.509 Token or Require Signing are also required. 	No. Mediator includes only one action in the effective policy.
10	Authorize User	Require HTTP Basic Authentication, Require WSS Username Token <i>or</i> Require WSS SAML Token.	No. Mediator includes only one action in the effective policy.

Evaluation Order	Action	Dependency	Can include multiple times in a policy?
11	Authorize Against Registered Consumers	Identify Consumer action.	No. Mediator includes only one action in the effective policy.
12	Validate Schema	None.	If at least one occurrence of the action is configured to validate requests, and at least one occurrence of the action is configured to validate responses, then Mediator includes in the effective policy an action to validate both requests and responses. Otherwise, an action is chosen which validates only requests or only responses (depending on the value of the Validate SOAP Messages parameter of the action).
13	Log Invocation	None.	No. Mediator includes only one action in the effective policy.
14	Monitor Service Performance	None.	Yes. Mediator includes all Monitor Service Performance actions in the effective policy.
15	Monitor Service Level Agreement	Identify Consumer action.	Yes. Mediator includes all Monitor Service Level Agreement actions in the effective policy.

Evaluation Order	Action	Dependency	Can include multiple times in a policy?
16	Throttling Traffic Optimization	Identify Consumer (if the Limit Traffic for Applications option is selected).	Yes. Mediator includes all Throttling Traffic Optimization actions in the effective policy.

Usage Cases for Identifying/Authenticating Consumers

When deciding which type of identifier to use to identify a consumer application, consider the following points:

- Whatever identifier you choose to identify a consumer application, it must be unique to the application. Identifiers that represent user names are often not suitable because the identified users might submit requests for multiple applications.
- Identifying applications by IP address or host name is often a suitable choice, however, it does create a dependency on the network infrastructure. If a consumer application moves to a new machine, or its IP address changes, you must update the identifiers in the application asset.
- Using X.509 certificates or a custom token that is extracted from the SOAP message itself (using an XPATH expression), is often the most trouble-free way to identify a consumer application.

Following are some common combinations of actions used to authenticate/identify consumers.

■ Scenario 1: Identify consumers by IP address or host name

- The simplest way to identify consumers is to use the Identify Consumer action and set its `Identify User Using` parameter to specify either a host name or an IP address (or a range of IP addresses).

■ Scenario 2: Authenticate consumers by HTTP authentication token

Use the following actions:

- Identify Consumer action, and set its `Identify User Using` parameter to HTTP Authentication Token (to identify consumers using the token derived from the HTTP header).
- Require HTTP Basic Authentication.
- Additionally, you can use one or both of the following:
 - Authorize User action (to authorize a list of users and/or groups registered in the Integration Server on which Mediator is running).

- Authorize Against Registered Consumers action (to authorize consumer applications against all Application assets registered as consumers for a service in CentraSite).
- **Scenario 3: Authenticate consumers by WS-Security authentication token**

Use the following actions:

 - Identify Consumer action, and set its `Identify User Using` parameter to `WS-Security Authentication Token` (to identify consumers using the token derived from the WSS Header).
 - Require WSS Username Token action.
 - Additionally, you can use one or both of the following:
 - Authorize User action (to authorize a list of users and/or groups registered in the Integration Server on which Mediator is running).
 - Authorize Against Registered Consumers action (to authorize consumer applications against all Application assets registered as consumers for a service in CentraSite).
- **Scenario 4: Authenticate consumers by WSS X.509 token**
 - Identify Consumer action, and set its `Identify User Using` parameter to `Consumer Certificate` (to identify consumers using the WSS X.509 token).
 - Require WSS X.509 Token action
 - Require SSL action.

Run-Time Actions Reference for Virtual Services

This section describes the following built-in run-time actions that you can include in run-time policies for virtual services:

- [Authorize Against Registered Consumers](#)
- [Authorize User](#)
- [Identify Consumer](#)
- [Log Invocation](#)
- [Monitor Service Performance](#)
- [Monitor Service Level Agreement](#)
- [Require Encryption](#)
- [Require HTTP Basic Authentication](#)
- [Require Signing](#)
- [Require SSL](#)

- [Require Timestamps](#)
- [Require WSS SAML Token](#)
- [Require WSS Username Token](#)
- [Require WSS X.509 Token](#)
- [Throttling Traffic Optimization](#)
- [Validate Schema](#)

Authorize Against Registered Consumers

Note: Dependency requirement: A policy that includes this action must also include the [Identify Consumer](#) action. However, if the [Identify Consumer](#) action is set to identify users via the **HTTP Authentication Token** option, then "Authorize Against Registered Consumers" should not be included in the policy.

Authorizes consumer applications against all consumer applications who are registered in CentraSite as consumers for the service.

Input Parameters

None.

Authorize User

Note: Dependency requirement: A policy that includes this action must also include *one* of the following: the [Require WSS SAML Token](#) action or the [Identify Consumer](#) action with one of the following options selected: "HTTP Authentication Token" or "WS-Security Authentication Token".

Authorizes consumers against a list of users and/or a list of groups registered in the Integration Server on which Mediator is running.

Input Parameters

Perform authorization
against list of users

Boolean. Authorizes consumers against a list of users who are registered in the Integration Server on which Mediator is running. Specify one or more users in the fields below this option.

Perform authorization
against list of groups

Boolean. Authorizes consumers against a list of groups who are registered in the Integration Server on which Mediator is running. Specify one or more groups in the fields below this option.

Note: By default, both of the input parameters are selected. If you de-select one of these parameters, the fields showing the list of users (or groups) is not displayed.

Identify Consumer

Mediator uses this action to identify consumer applications based on the kind of consumer identifier (IP address, HTTP authorization token, and so on.) you specify. Alternatively, this action provides an option to allow anonymous users to access the assets.

Input Parameters

Anonymous Usage Allowed

Boolean. Specifies whether to allow all users to access the asset, without restriction.

Value	Description
False	<i>Default.</i> Allows only the users specified in the <code>Identify User Using</code> parameter to access the assets.
True	Allow all users to access the asset. In this case, do not configure the <code>Identify User Using</code> parameter.

Identify User Using

String. Specifies the kind of consumer identifier that the action will use to identify consumer applications.

Value	Description
IP Address	Identifies one or more consumer applications based on their originating IP addresses.
Host Name	Identifies consumer applications based on a host name.
HTTP Authentication Token	Uses HTTP Basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header. Mediator authorizes the credentials against the list of consumers available in the Integration Server on which

Mediator is running. This type of consumer authentication is referred to as “preemptive authentication”. If you want to use “preemptive authentication”, you should also include the action [Require HTTP Basic Authentication](#) in the policy.

If you choose to omit “Require HTTP Basic Authentication”, the client will be presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. This type of consumer authentication is referred to as “non-preemptive authentication”. For more information, see ["Require HTTP Basic Authentication" on page 373](#).

Note: If you select the value HTTP Authentication Token, do not include the Authorize Against Registered Consumers action in the policy. This is an invalid combination.

WS-Security Authentication Token	Validate user names and passwords that are transmitted in the SOAP message header in the WSS Username Token. If you select this value, you should also include the action Require WSS Username Token in the policy.
Custom Identification	Validates consumer applications based on an XML element (represented by an XPath expression).
Consumer Certificate	Identifies consumer applications based on information in a WSS X.509 certificate. If you select this value, you should also include the action Require WSS X.509 Token or the action Require Signing in the policy.
Client Certificate for SSL Connectivity	Validates the client's certificate that the consumer application submits to the asset in CentraSite. The client certificate that is used to identify the consumer is supplied by the client to

the Mediator during the SSL handshake over the transport layer. In order to identify consumers by transport-level certificates, the run-time communication between the client and the Mediator must be over HTTPS and the client must pass a valid certificate.

To use this option, the following prerequisites must be met:

- In Integration Server, create a keystore and truststore, as described in *webMethods Integration Server Administrator's Guide*.
- In Integration Server, create an HTTPS port, as described in *webMethods Integration Server Administrator's Guide*.
- Configure Mediator by setting the IS Keystore and IS Truststore parameters, as described in *Administering webMethods Mediator*.
- Configure Mediator by setting the HTTPS Ports Configuration parameter, as described in *Administering webMethods Mediator*.

When deciding which type of identifier to use to identify a consumer application, consider the following points:

- Whatever identifier you choose to identify a consumer application, it must be unique to the application. Identifiers that represent user names are often not suitable because the identified users might submit requests for multiple applications.
- Identifying applications by IP address or host name is often a suitable choice, however, it does create a dependency on the network infrastructure. If a consumer application moves to a new machine, or its IP address changes, you must update the identifiers in the application asset.
- Using X.509 certificates or a custom token that is extracted from the SOAP or XML message itself (using an XPATH expression), is often the most trouble-free way to identify a consumer application.

Log Invocation

Logs request/response payloads. You can specify the log destination and the logging frequency. This action also logs other information about the requests/responses, such as the service name, operation name, the Integration Server user, a timestamp, and the response time.

Note: You can include this action multiple times in a policy.

Input Parameters

Log the
Following
Payloads

String. Optional. Specifies whether to log all request payloads, all response payloads, or both.

<u>Value</u>	<u>Description</u>
Request	Log all request payloads.
Response	Log all response payloads.

Log
Generation
Frequency

String. Specifies how frequently to log the payload.

<u>Value</u>	<u>Description</u>
Always	Log all requests and/or responses.
On Success	Log only the successful responses and/or requests.
On Failure	Log only the failed requests and/or responses.

Send Data To

String. Specifies where to log the payload.

Important: Ensure that Mediator is configured to log the payloads to the destination(s) you specify here. For details about alerts and transaction logging, see *Administering webMethods Mediator*.

<u>Value</u>	<u>Description</u>
CentraSite	<p>Logs the payloads in the virtual service's Events profile in CentraSite.</p> <p>Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to Solutions > Mediator > Administration > CentraSite Communication). For the procedure, see <i>Administering webMethods Mediator</i>.</p>

- Local Log Logs the payloads in the server log of the Integration Server on which Mediator is running.
- Also choose a value in the Log Level field:
- Info: Logs error-level, warning-level, and informational-level alerts.
 - Warn: Logs error-level and warning-level alerts.
 - Error: Logs only error-level alerts.

Important: The Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to **Settings > Logging > Server Logger**).

- SNMP Logs the payloads in CentraSite's SNMP server or a third-party SNMP server.

Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > SNMP**). For the procedure, see *Administering webMethods Mediator*.

- Email Sends the payloads to an SMTP email server, which sends them to the email address(es) you specify here. Mediator sends the payloads as email attachments that are compressed using gzip data compression. To specify multiple addresses, use the plus button to add rows.

Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > Email**). For the procedure, see *Administering webMethods Mediator*.

- Audit Log Logs the payload to the Integration Server audit logger. For more information about logging, see the *webMethods Audit Logging Guide*.

Note: If you expect a high volume of events in your system, it is recommended that you select the Audit Log destination for this action.

EDA	<p>Sends data about run-time events and KPI metrics to a database or a messaging server such as webMethods Universal Messaging depending on the destination that you configure in Integration Server.</p> <p>Prerequisite You must configure the EDA destination in Integration Server on the Solutions > Mediator > Administration > EDA Configuration page. For details, see <i>Administering webMethods Mediator</i>.</p>
-----	---

Monitor Service Performance

This action monitors a user-specified set of run-time performance conditions for a virtual service, and sends alerts to a specified destination when the performance conditions are violated. You can include this action multiple times in a single policy.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), Mediator sends an alert as soon as the performance condition is violated, without having to wait until the end of the metrics tracking interval. You can choose whether to send an alert only once during the interval, or every time the violation occurs during the interval. (Mediator will send another alert the next time a condition is violated during a subsequent interval.) For information about the metrics tracking interval, see "[The Metrics Tracking Interval](#)" on page 332.

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), Mediator aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

This action does not include metrics for failed invocations.

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see *Administering webMethods Mediator*.

Input Parameters

Action Configuration parameters	Specify one or more conditions to monitor. To do this, specify a metric, operator, and a value for each metric. To specify multiple conditions, use the plus button to add multiple rows. If multiple parameters are used, they are connected by the AND operator.
Name	<i>String Array</i> . The metrics to monitor.

	<u>Value</u>	<u>Description</u>
	Availability	Indicates whether the service was available to the specified consumers in the current interval.
	Average Response Time	The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment Mediator receives the request until the moment it returns the response to the caller.
	Concurrent Connections	Indicates whether the service was accessed by multiple consumers at the same time in the current interval.
	Fault Count	The number of faults returned in the current interval.
	Maximum Response Time	The maximum amount of time to respond to a request in the current interval.
	Minimum Response Time	The minimum amount of time to respond to a request in the current interval.
	Successful Request Count	The number of successful requests in the current interval.
	Total Request Count	The total number of requests (successful and unsuccessful) in the current interval.
Operator	<i>String Array.</i> Choose an appropriate operator.	
Value	<i>String Array.</i> Specify an appropriate value.	
Alert parameters	<i>Object.</i> Specify the following parameters for the alerts that will report on the conditions:	
Alert Interval	<i>Number.</i> The time period (in minutes) in which to monitor performance before sending an alert if a condition is violated.	

For information about the metrics tracking interval, see "[The Metrics Tracking Interval](#)" on page 332.

Alert Frequency *String*. Specifies how frequently to issue alerts for the counter-based metrics (Total Request Count, Success Count, Fault Count).

Value	Description
Every Time	Issue an alert every time one of the specified conditions is violated.
Only Once	Issue an alert only the first time one of the specified conditions is violated.

Reply to Destination *String*. Specifies where to send the alerts.

Important: Ensure that Mediator is configured to send event notifications to the destination(s) you specify here. For details, see *Administering webMethods Mediator*

Value	Description
CentraSite	<p>Sends the alerts to the virtual service's Events profile in CentraSite.</p> <p>Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to Solutions > Mediator > Administration > CentraSite Communication). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
Local Log	<p>Sends the alerts to the server log of the Integration Server on which Mediator is running.</p> <p>Also choose a value in the <code>Log Level</code> field:</p> <ul style="list-style-type: none"> ■ Info: Logs error-level, warning-level, and informational-level alerts. ■ Warn: Logs error-level and warning-level alerts. ■ Error: Logs only error-level alerts.

Important: The Integration Server Administrator's logging level for Mediator should

match the logging level specified for this action (go to **Settings > Logging > Server Logger**).

SNMP	<p>Sends the alerts to CentraSite's SNMP server or a third-party SNMP server.</p> <p>Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
Email	<p>Sends the alerts to an SMTP email server, which sends them to the email address(es) you specify here. To specify multiple addresses, use the plus button to add rows.</p> <p>Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
EDA	<p>Sends data about run-time events and KPI metrics to a database or a messaging server such as webMethods Universal Messaging depending on the destination that you configure in Integration Server.</p> <p>Prerequisite You must configure the EDA destination in Integration Server on the Solutions > Mediator > Administration > EDA Configuration page. For details, see <i>Administering webMethods Mediator</i>.</p>

Alert Message *String. Optional.* Specify a text message to include in the alert.

Monitor Service Level Agreement

Note: Dependency requirement: A policy that includes this action must also include the [Identify Consumer](#) action.

This action is similar to the Monitor Service Performance action. Both actions can monitor the same set of run-time performance conditions for a virtual service, and then send alerts when the performance conditions are violated. This action is different because it enables you to monitor run-time performance for *one or more specified consumers*. You can include this action multiple times in a single policy.

You can configure this action to define a *Service Level Agreement (SLA)*, which is a set of conditions that defines the level of performance that a consumer should expect from a service. You can use this action to identify whether a service's threshold rules are met or exceeded. For example, you might define an agreement with a particular consumer that sends an alert to the consumer if responses are not sent within a certain maximum response time. You can configure SLAs for each virtual service/consumer application combination.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), Mediator sends an alert as soon as the performance condition is violated, without having to wait until the end of the metrics tracking interval. You can choose whether to send an alert only once during the interval, or every time the violation occurs during the interval. (Mediator will send another alert the next time a condition is violated during a subsequent interval.) For information about the metrics tracking interval, see "[The Metrics Tracking Interval](#)" on page 332.

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), Mediator aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

This action does not include metrics for failed invocations.

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see *Administering webMethods Mediator*.

Input Parameters

Action Configuration parameters Specify one or more conditions to monitor. To do this, specify a metric, operator, and value for each metric. To specify multiple conditions, use the plus button to add multiple rows. If multiple parameters are used, they are connected by the AND operator.

NameOperator *String Array*. The metrics to monitor.

Value	Description
Availability	Indicates whether the service was available to the specified consumers in the current interval.
Average Response Time	The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment

	Mediator receives the request until the moment it returns the response to the caller.
Fault Count	Indicates the number of faults returned in the current interval.
Maximum Response Time	The maximum amount of time to respond to a request in the current interval.
Minimum Response Time	The minimum amount of time to respond to a request in the current interval.
Successful Request Count	The number of successful requests in the current interval.
Total Request Count	The total number of requests (successful and unsuccessful) in the current interval.
	<i>String Array.</i> Choose an appropriate operator.
Value	<i>String Array</i> Specify an appropriate value.
Alert for Consumer Applications	<i>Object Array.</i> Specify the Application asset(s) to which this Service Level Agreement will apply. To specify multiple Application assets, use the plus button to add multiple rows.
Alert parameters	<i>Object.</i> Specify the following parameters for the alerts that will report on the Service Level Agreement conditions:
Alert Interval	<i>Number.</i> The time period (in minutes) in which to monitor performance before sending an alert if a condition is violated. For information about the metrics tracking interval, see " The Metrics Tracking Interval " on page 332.
Alert Frequency	<i>String.</i> Specifies how frequently to issue alerts for the counter-based metrics (Total Request Count, Success Count, Fault Count).

	<u>Value</u>	<u>Description</u>
	Every Time	Issue an alert every time one of the specified conditions is violated.
	Only Once	Issue an alert only the first time one of the specified conditions is violated.
Rule Expiration Date		<i>String</i> . Specifies the date on which this Service Monitoring Performance action expires, in format MM/DD/YYYY.
Reply to Destination		<i>String</i> . Specifies where to log the alert.
		Important: Ensure that Mediator is configured to send event notifications to the destination(s) you specify here. For details, see <i>Administering webMethods Mediator</i> .

	<u>Value</u>	<u>Description</u>
	CentraSite	Sends the alerts to the virtual service's Events profile in CentraSite. Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to Solutions > Mediator > Administration > CentraSite Communication). For the procedure, see <i>Administering webMethods Mediator</i> .
	Local Log	Sends the alerts to the server log of the Integration Server on which Mediator is running. Also choose a value in the Log Level field: ■ Info: Logs error-level, warning-level, and informational-level alerts.

- **Warn:** Logs error-level and warning-level alerts.
- **Error:** Logs only error-level alerts.

Important! Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to **Settings > Logging > Server Logger**).

SNMP

Sends the alerts to CentraSite's SNMP server or a third-party SNMP server.

Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > Email**). For the procedure, see *Administering webMethods Mediator*.

Email

Sends the alerts to an SMTP email server, which sends them to the email address(es) you specify here. To specify multiple addresses, use the plus button to add rows.

Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > Email**). For the procedure, see *Administering webMethods Mediator*.

EDA

Sends data about run-time events and KPI metrics to a database or a messaging server such as webMethods Universal Messaging depending on the

destination that you configure in Integration Server.

Prerequisite You must configure the EDA destination in Integration Server on the **Solutions > Mediator > Administration > EDA Configuration** page. For details, see *Administering webMethods Mediator*.

Alert Message

String. Optional. Specify a text message to include in the alert.

Require Encryption

Requires that a request's XML element (which is represented by an XPath expression) be encrypted. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services.

Prerequisites

1. Configure Integration Server: Set up keystores and truststores in Integration Server, as described in *webMethods Integration Server Administrator's Guide*.
2. Configure Mediator: In the Integration Server Administrator, navigate to **Solutions > Mediator > Administration > General** and complete the IS Keystore Name, IS Truststore Name and Alias (signing) fields, as described in *Administering webMethods Mediator*.

When this policy action is set for the virtual service, Mediator provides decryption of incoming requests and encryption of outgoing responses. Mediator can encrypt and decrypt only individual elements in the SOAP message body that are defined by the XPath expressions configured for the policy action. Mediator requires that requests contain the encrypted elements that match those in the XPath expression. You must encrypt the entire element, not just the data between the element tags. Mediator rejects requests if the element name is not encrypted.

Important: Do not encrypt the entire SOAP body because a SOAP request without an element will appear to Mediator to be malformed.

Mediator attempts to encrypt the response elements that match the XPath expressions with those defined for the policy. If the response does not have any elements that match the XPath expression, Mediator will not encrypt the response before sending. If the XPath expression resolves a portion of the response message, but Mediator cannot locate a certificate to encrypt the response, then Mediator sends a SOAP fault exception to the consumer and a Policy Violation event notification to CentraSite.

How Mediator Encrypts Responses

The Require Encryption action encrypts the response back to the client by dynamically setting a public key alias at run time. Mediator determines the public key alias as follows:

1. If Mediator can access the X.509 certificate of the client (based on the incoming request signature), it will use “useReqSigCert” as the public key alias.
OR
2. If the Identify Consumer action is present in the policy (and it successfully identifies a consumer application), then Mediator will look for a public key alias with that consumer name in the “IS Keystore Name” property. The “IS Keystore Name” property is specified in the Integration Server Administrator, under **Solutions > Mediator > Administration > General**. This property should be set to an Integration Server keystore that Mediator will use.

For an Identify Consumer action that allows for anonymous usage, Mediator does *not* require a consumer name in order to send encrypted responses. In this case, Mediator can use one of the following to encrypt the response in the following order, depending on what is present in the security element:

- A signing certificate.
- Consumer name.
- WSS username, SAML token, or X.509 certificate.
- HTTP authorized user.

OR

3. If Mediator can determine the current IS user from the request (that is, if an Integration Server WS-Stack determined that Subject is present), then the first principal in that subject is used.

OR

4. If the above steps all fail, then Mediator will use either the WS-Security username token or the HTTP Basic-Auth user name value. There should be a public key entry with the same name as the identified username.

Note: You can include this action multiple times in a single policy.

Input Parameters

Namespace *String. Optional.* Namespace of the element required to be encrypted.

Note: Enter the namespace prefix in the following format:
xmlns:<prefix-name>. For example: xmlns:soapenv.

The generated XPath element in the policy should look similar to this:

```
<sp:SignedElements
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-security
policy/200702">
<sp:XPath
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
//soapenv:Body</sp:XPath>
</sp:SignedElements>
```

Element	<i>String</i> . An XPath expression that represents the XML element
Required to be	that is required to be encrypted.
Encrypted	

Require HTTP Basic Authentication

This action uses HTTP Basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header. Mediator authorizes the credentials against the list of consumers available in the Integration Server on which Mediator is running. This type of consumer authentication is referred to as “preemptive authentication”. If you want to perform “preemptive authentication”, a policy that includes this action must also include the Identify Consumer action.

If the user/password value in the Authorization header cannot be authenticated as a valid Integration Server user (or if the Authorization header is not present in the request), a 500 SOAP fault is returned, and the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. This type of consumer authentication is referred to as “non-preemptive authentication”. If the client does not successfully respond to the challenge, a 401 “WWW-Authenticate: Basic” response is returned and the invocation is not routed to the policy engine. As a result, no events are recorded for that invocation, and its key performance indicator (KPI) data are not included in the performance metrics.

If you choose to omit the “Require HTTP Basic Authentication” action (and regardless of whether an Authorization header is present in the request or not), then:

- Mediator forwards the request to the native service, without attempting to authenticate the request.
- The native service returns a 401 “WWW-Authenticate: Basic” response, which Mediator will forward to the client; the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated.

In the case where a consumer sends a request with transport credentials (HTTP Basic authentication) and message credentials (WSS Username or WSS X.509 token), the message credentials take precedence over the transport credentials when Integration Server determines which credentials it should use for the session. For more information, see ["Require WSS Username Token" on page 377](#) and ["Require WSS X.509 Token" on page 378](#). In addition, you must ensure that the service consumer that connects to the virtual service has an Integration Server user account.

Note: Do not include the “Require HTTP Basic Authentication” action in a virtual service's run-time policy if you selected the **OAuth2** option in the virtual service's Routing Protocol step.

Input Parameters

Note: This input parameter is not available in Mediator versions prior to 9.0.

Authenticate
Credentials

Required. Authorizes consumers against the list of consumers available in the Integration Server on which Mediator is running.

Require Signing

This action requires that a request's XML element (which is represented by an XPath expression) be signed. This action supports WS-SecurityPolicy 1.2.

Prerequisites

1. Configure Integration Server: Set up keystores and truststores in Integration Server, as described in *webMethods Integration Server Administrator's Guide*.
2. Configure Mediator: In the Integration Server Administrator, navigate to **Solutions > Mediator > Administration > General** and complete the IS Keystore Name, IS Truststore Name, and Alias (signing) fields, as described in *Administering webMethods Mediator*. Mediator uses the signing alias specified in the Alias (signing) field to sign the response.

When this action is set for the virtual service, Mediator validates that the requests are properly signed, and provides signing for responses. Mediator provides support both for signing an entire SOAP message body or individual elements of the SOAP message body.

Mediator uses a digital signature element in the security header to verify that all elements matching the XPath expression were signed. If the request contains elements that were not signed or no signature is present, then Mediator rejects the request.

Note: Keep the following in mind:

1. You must map the public certificate of the key used to sign the request to an Integration Server user. If the certificate is not mapped, Mediator returns a SOAP fault to the caller.
2. You can include this action multiple times in a policy.

Input Parameters

Namespace *String. Optional.* Namespace of the element required to be signed.

Note: Enter the namespace prefix in the following format:
`xmlns:<prefix-name>`. For example: `xmlns:soapenv`.

The generated XPath element in the policy should look similar to this:

```
<sp:SignedElements
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-security
policy/200702">
<sp:XPath
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
//soapenv:Body</sp:XPath>
</sp:SignedElements>
```

Element Required to be Signed *String.* An XPath expression that represents the XML element that is required to be signed.

Require SSL

Requires that requests be sent via SSL client certificates. This action supports WS-SecurityPolicy 1.2 and can be used for both SOAP and REST services.

When this action is set for the virtual service, Mediator ensures that requests are sent to the server using the HTTPS protocol (SSL). The action also specifies whether the client certificate is required. This allows Mediator to verify the client sending the request. If the policy requires the client certificate, but it is not presented, Mediator rejects the message.

When a client certificate is required, the Integration Server HTTPS port should be configured to request or require a client certificate.

Input Parameters

Client Certificate Required *Boolean.* Specifies whether client certificates are required for the purposes of:

- Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests.
- Signing SOAP responses or encrypting SOAP responses.

Value	Description
Yes	Require client certificates.

No Default. Do not require client certificates.

Require Timestamps

Note: Dependency requirement: A policy that includes this action must also include *any* one of the following actions: [Require Signing](#), [Require Encryption](#).

When this policy action is set for the virtual service, Mediator requires that timestamps be included in the request header. Mediator checks the timestamp value against the current time to ensure that the request is not an old message. This serves to protect your system against attempts at message tampering, such as replay attacks. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services.

Mediator rejects the request if either of the following happens:

- Mediator receives a timestamp that exceeds the time defined by the timestamp element.
- A timestamp element is not included in the request.

Input Parameters

None.

Require WSS SAML Token

When this action is set for a virtual service, Mediator uses a WSS Security Assertion Markup Language (SAML) assertion token to validate service consumers. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services.

For more information about configuring your system for SAML token processing, see *Administering webMethods Mediator*.

Input Parameters

SAML Subject Confirmation *String*. Select one of the following SAML subject confirmation methods:

Value	Description
Holder of Key	<i>Default</i> . Select this option if consumers use the SAML V1.1 or V2.0 Holder-of-Key Web Browser SSO Profile, which allows for transport of holder-of-key assertions. In this scenario, the consumer presents a holder-of-key SAML assertion acquired from its preferred identity provider to

access a web-based resource at a service provider.

If you select `Holder of Key`, Mediator also implicitly selects the “timestamp” and “signing” assertions to the virtual service definition (VSD). Thus, you should not add the “Require Timestamps” and “Require Signing” policy actions to a virtual service if the “Require WSS SAML Token” action is already applied.

`Bearer`

Select this option if consumers use SAML V1.1 Bearer token authentication, in which a Bearer token mechanism relies upon bearer semantics as a means by which the consumer conveys to Mediator the sender's identity.

If you select `Bearer`, the “timestamp” and “signing” assertions will be added to the virtual service definition (VSD).

Note If consumers use SAML 2.0 Sender-Vouches tokens, configure your system as described in *Administering webMethods Mediator*.

`SAML Version` *String*. Specifies the WSS SAML Token version to use: 1.1 or 2.0.

Require WSS Username Token

Note: Dependency requirement: A policy that includes this action must also include the Identify Consumer action.

When this policy action is set for the virtual service, Mediator uses WS-SecurityPolicy authentication to validate user names and passwords that are transmitted in the SOAP message header for the WSS Username token. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services.

In the case where a consumer is sending a request with both transport credentials (HTTP basic authentication) and message credentials (WSS Username or X.509 token), the message credentials take precedent over the transport credentials when Integration Server is determining which credentials it should use for the session. For more information, see ["Require HTTP Basic Authentication" on page 373](#).

Mediator rejects requests that do not include the username token and password of an Integration Server user. Mediator only supports clear text passwords with this kind of authentication

Input Parameters

None.

Require WSS X.509 Token

Note: Dependency requirement: A policy that includes this action must also include the Identify Consumer action.

Identifies consumers based on a WSS X.509 token. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services.

In the case where a consumer is sending a request with both transport credentials (HTTP Basic authentication) and message credentials (WSS X.509 token or WSS Username), the message credentials take precedence over the transport credentials when Integration Server is determining which credentials it should use for the session. For more information, see ["Require HTTP Basic Authentication" on page 373](#). In addition, you must ensure that the service consumer that connects to the virtual service has an Integration Server user account.

Input Parameters

None.

Throttling Traffic Optimization

Note: Keep the following in mind:

1. This action is not available in Mediator versions below 9.0.
2. Dependency requirement: A policy that includes this action must also include the Identify Consumer action if the `Limit Traffic for Applications` option is selected.

This action limits the number of service invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated.

Reasons for limiting the service invocation traffic include:

- To avoid overloading the back-end services and their infrastructure.
- To limit specific consumers in terms of resource usage (that is, you can use the "Monitor Service Level Agreement" action to monitor performance conditions for a particular consumer, together with "Throttling Traffic Optimization" to limit the resource usage).
- To shield vulnerable servers, services, and even specific operations.

- For service consumption metering (billable pay-per-use services).

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see *Administering webMethods Mediator*.

Input Parameters

Soft Limit	<p><i>Number. Optional.</i> Specifies the maximum number of invocations allowed per Interval before issuing an alert. Reaching the soft limit will not affect further processing of requests (until the Hard Limit is reached).</p> <p>Note: The limit is reached when the total number of invocations coming from <i>all</i> the consumer applications (specified in the Limit Traffic for Applications field) reaches the limit. Soft Limit is computed in an asynchronous manner; thus when multiple requests are made at the same time, it may be possible that the Soft Limit alert will not be strictly accurate.</p>
Hard Limit	<p><i>Number. Required.</i> Specifies the maximum number of invocations allowed per alert interval before stopping the processing of further requests and issuing an alert. Typically, this number should be higher than the soft limit.</p> <p>Note: The limit is reached when the total number of invocations coming from <i>all</i> the consumer applications (specified in the Limit Traffic for Applications field) reaches the limit. Hard Limit is computed in an asynchronous manner; thus when multiple requests are made at the same time, it may be possible that the Hard Limit alert will not be strictly accurate.</p>
Limit Traffic for Applications	<p><i>String.</i> Specifies the consumer application(s) that this action applies to. To specify multiple consumer applications, use the plus button to add rows, or select Any Consumer to apply this action to any consumer application.</p>
Interval	<p><i>Number.</i> Specifies the amount of time for the soft limit and hard limit to be reached.</p>
Frequency	<p><i>String.</i> Specifies how frequently to issue alerts.</p>

Reply To
Destination

Value	Description
Every Time	Issue an alert every time the specified condition is violated.
Only Once	Issue an alert only the first time the specified condition is violated.

String. Optional. Specifies where to log the alerts.

Important: Ensure that Mediator is configured to send event notifications to the destination(s) you specify here. For details, see *Administering webMethods Mediator*.

Value	Description
CentraSite	<p>Sends the alerts to the virtual service's Events profile in CentraSite.</p> <p>Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to Solutions > Mediator > Administration > CentraSite Communication). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
Local Log	<p>Sends the alerts to the server log of the Integration Server on which Mediator is running.</p> <p>Also choose a value in the Log Level field:</p> <ul style="list-style-type: none"> ■ Info: Logs error-level, warning-level, and informational-level alerts. ■ Warn: Logs error-level and warning-level alerts. ■ Error: Logs only error-level alerts.

Important: Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to **Settings > Logging > Server Logger**).

	SNMP	<p>Sends the alerts to CentraSite's SNMP server or a third-party SNMP server.</p> <p>Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
	Email	<p>Sends the alerts to an SMTP email server, which sends them to the email address(es) you specify here. To specify multiple addresses, use the plus button to add rows.</p> <p>Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
	EDA	<p>Sends data about run-time events and KPI metrics to a database or a messaging server such as webMethods Universal Messaging depending on the destination that you configure in Integration Server.</p> <p>Prerequisite You must configure the EDA destination in Integration Server on the Solutions > Mediator > Administration > EDA Configuration page. For details, see <i>Administering webMethods Mediator</i>.</p>
Alert Message for Soft Limit		<i>String. Optional.</i> Specify a text message to include in the soft limit alert.
Alert Message for Hard Limit		<i>String. Optional.</i> Specify a text message to include in the hard limit alert.

Validate Schema

This action validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Mediator can enforce this policy action for messages sent between services. When this policy is set for the virtual service, Mediator validates XML request messages, response messages, or both, against the XML schema referenced in the WSDL.

Input Parameters

Validate SOAP
Message(s)

Object. Validates request and/or response messages. You may select both Request and Response.

Value	Description
Request	Validate all requests.
Response	Validate all responses.

Important: Be aware that Mediator does not remove `wsu:Id` attributes that may have been added to a request by a consumer as a result of security operations against request elements (that is, signatures and encryptions). In this case, to avoid schema validation failures you would have to add a Request Handling step to the virtual service so that the requests are passed to an XSL transformation file that removes the `wsu:Id` attribute.

Built-In Run-Time Actions Reference for APIs

This section describes the built-in run-time actions that you can include in run-time governance rules for APIs. You use these actions only when you are using the CentraSite Business UI to create run-time policies for APIs. The content is organized under the following sections:

Instructions throughout the remainder of this guide use the term “API” when referring to the Virtual Services, Virtual XML Services and Virtual REST Services; and the term “client” when referring to the consumer applications in general.

Summary of the Run-Time Actions

You can include the following kinds of built-in run-time actions in the run-time governance rules for APIs:

Request Handling Actions

Request Handling is the process of receiving and transforming the incoming message from a client into the custom format as expected by the native API.

Require HTTP / HTTPS	Specifies the protocol (HTTP or HTTPS) and SOAP format (for a SOAP-based API) to be used to accept and process the requests.
Require JMS	Specifies the JMS protocol to be used for the API to accept and process the requests.
Request Transformation	Invokes an XSL transformation in the SOAP request before it is submitted to the native API.
Invoke webMethods Integration Server	Invokes a webMethods Integration Server service to pre-process the request before it is submitted to the native API.
Enable REST Support	Enables REST support for an existing SOAP based API by exposing the API both as a SOAP based API and a REST based API.
Set Media Type	Specifies the content type for a REST request received from a client if the content type header is not specified.

Policy Enforcement Actions

Policy Enforcement is the process of enforcing the adherence to real-time policy compliance identifying/authenticating, monitoring, auditing, and measuring and collecting result statistics for an API.

Mediator provides the following categories of policy enforcement actions:

- [Authentication Actions](#)
- [JMS Routing Actions](#)
- [Logging and Monitoring Actions](#)
- [Routing Actions](#)
- [Security Actions](#)
- [Traffic Management Action](#)
- [Validate Schema](#)

Authentication Actions

Authentication actions verify that the API client has the proper credentials to access an API.

HTTP Basic Authentication	Uses HTTP basic authentication to verify the client's authentication credentials contained in the request's
----------------------------------	---

Authorization header against the Integration Server's user account.

NTLM Authentication Uses NTLM authentication to verify the client's authentication credentials contained in the request's Authorization header against the Integration Server's user account.

OAuth2 Authentication Uses OAuth2 authentication to verify the client's authentication credentials contained in the request's Authorization header against the Integration Server's user account.

JMS Routing Actions

JMS Routing actions route the incoming message to an API over JMS. For example, to a JMS queue where an API can then retrieve the message asynchronously.

JMS Routing Rule Specifies a JMS queue to which the Mediator is to submit the request, and the destination to which the native API is to return the response.

Set Message Properties Specifies JMS message properties to authenticate client requests before submitting to the native APIs.

Set JMS Headers Specifies JMS headers to authenticate client requests before submitting to the native APIs.

Logging and Monitoring Actions

Logging and Monitoring actions monitor and collect information about the number of messages that were processed successfully or failed, the average execution time of message processing, and the number of alerts associated with an API.

Log Invocation Logs request/response payloads to a destination you specify.

Monitor Service Level Agreement Specifies a Service Level Agreement (SLA), which is set of conditions that define the level of performance that a specified client should expect from an API.

Monitor Service Performance This action provides the same functionality as Monitor Service Level Agreement but this action is different because it enables you to monitor the API's run-time performance for all clients. This action monitors a user-specified set of run-time performance conditions for an

API, and sends alerts to a specified destination when these performance conditions are violated.

Routing Actions

Routing actions route the incoming message (for example, directly to the API, or routed according to the routing rules, or routed to a pool of servers for the purpose of load balancing and failover handling).

Straight Through Routing	Routes the requests directly to a native endpoint that you specify.
Context Based Routing	Route requests to different endpoints based on specific values that appear in the request message.
Content Based Routing	Route requests to different endpoints based on specific criteria that you specify.
Load Balancing and Failover Routing	Routes the requests across multiple endpoints.
Set Custom Headers	Specifies the HTTP headers to process the requests.

Security Actions

Security actions provide client validation (through WSS X.509 certificates, WSS username tokens, and so on), confidentiality (through encryption) and integrity (through signatures) for request and response messages.

For the client validation, Mediator maintains a list of consumer applications specified in CentraSite that are authorized to access the API published to Mediator. Mediator synchronizes this list of consumer applications through a manual process initiated from CentraSite.

Generally speaking there are two different lists of consumers in the Mediator:

- **List of Registered Consumers**

List of users and consumer applications (represented as Application assets) who are registered as consumers for the API in CentraSite, and available in the Mediator.

- **List of Global Consumers**

List of all users and consumer applications (represented as consumers) available in the Mediator.

Mediator provides “Evaluate” actions that you can include in a message flow to identify and/or validate clients, and then configure their parameters to suit your needs. You use these “Evaluate” actions to perform the following actions:

- Identify the clients who are trying to access the APIs (through IP address or hostname).
- Validate the client's credentials.

Evaluate Client Certificate for SSL Connectivity

Mediator validates the client's certificate that the client submits to the API in CentraSite. The client certificate that is used to identify the client is supplied by the client to the Mediator during the SSL handshake over the transport layer.

Evaluate Hostname

- Mediator will try to identify the client against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available consumers in Mediator).
- Mediator will try to validate the client's hostname against the specified list of consumers in the Integration Server on which Mediator is running.

Evaluate HTTP Basic Authentication

- Mediator will try to identify the client against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available consumers in Mediator).
- Mediator will try to validate the client's authentication credentials contained in the request's Authorization header against the specified list of consumers in the Integration Server on which Mediator is running.

Evaluate IP Address

- Mediator will try to identify the client against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available consumers in Mediator).
- Mediator will try to validate the client's IP address against the specified list of consumers in the Integration Server on which Mediator is running.

Evaluate KerberosToken

Mediator will try to authenticate the client based on the Kerberos token and the authenticated client principal name is verified with the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available consumers in Mediator).

Evaluate OAuth2 Token

- Mediator will try to identify the client against either the Registered Consumers list (the list of registered

	<p>consumers in Mediator) or the Global Consumers list (the list of available consumers in Mediator).</p> <ul style="list-style-type: none">■ Mediator will try to validate the client's OAuth access token against the specified list of consumers in the Integration Server on which Mediator is running.
Evaluate WSS Username Token	<p><i>Applicable only for SOAP APIs.</i></p> <ul style="list-style-type: none">■ Mediator will try to identify the client against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available consumers in Mediator).■ Mediator will try to validate the client's WSS username token against the specified list of consumers in the Integration Server on which Mediator is running.
Evaluate WSS X.509 Certificate	<p><i>Applicable only for SOAP APIs.</i></p> <ul style="list-style-type: none">■ Mediator will try to identify the client against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available consumers in Mediator).■ Mediator will try to validate the client's WSS X.509 token against the specified list of consumers in the Integration Server on which Mediator is running.
Evaluate XPath Expression	<ul style="list-style-type: none">■ Mediator will try to identify the client against either the Registered Consumers list (the list of registered consumers in Mediator) or the Global Consumers list (the list of available consumers in Mediator).■ Mediator will try to validate the client's XPath expression against the specified list of consumers in the Integration Server on which Mediator is running.
Require Encryption	<p><i>Applicable only for SOAP APIs.</i></p> <p>Requires that a request's XML element (which is represented by an XPath expression) be encrypted.</p>
Require Signing	<p><i>Applicable only for SOAP APIs.</i></p> <p>Requires that a request's XML element (which is represented by an XPath expression) be signed.</p>
Require SSL	<p><i>Applicable only for SOAP APIs.</i></p> <p>Requires that requests be sent via SSL client certificates.</p>

Require Timestamps *Applicable only for SOAP APIs.*

Requires that timestamps be included in the request header. Mediator checks the timestamp value against the current time to ensure that the request is not an old message. This serves to protect your system against attempts at message tampering, such as replay attacks.

Require WSS SAML Token *Applicable only for SOAP APIs.*

Uses a WSS Security Assertion Markup Language (SAML) assertion token to validate API clients.

Traffic Management Action

Throttling Traffic Optimization

Limits the number of service invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated. You can use this action to avoid overloading the back-end services and their infrastructure, to limit specific clients in terms of resource usage, and so on.

Service Result Cache Enables caching of the results of SOAP and REST API invocations.

Validation Action

Validate Schema Validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Response Handling Actions

Response Handling is the process of transforming the response message coming from the native API into the custom format as expected by the client.

Response Transformation Invokes an XSL transformation in the response payloads from XML format to the format required by the client.

Invoke webMethods Integration Server Invokes a webMethods Integration Server service to process the response from the native API before it is returned to the client.

Set Media Type Specifies the content type for a REST response to the client if the content type header is not specified.

Error Handling Action

Error Handling is the process of passing an exception message which has been issued as a result of a run-time error to take any necessary actions.

Conditional Error Processing Returns a custom error message (and/or the native provider's service fault content) to the client when the native provider returns a service fault.

The `watt.server.auth.skipForMediator` Property

This property specifies whether Integration Server authenticates requests for Mediator. You must set this property to true.

No request to Mediator should be authenticated by Integration Server. Instead, authentication should be handled by Mediator. Thus, to enable Mediator to authenticate requests, you must set `skipForMediator` to true (by default, it is false).

When this parameter is set to true, Integration Server skips authentication for Mediator requests and allows the user credentials (of any type) to pass through so that Mediator can authenticate them. If you change the setting of this parameter, you must restart Integration Server for the changes to take effect.

To set `skipForMediator` to true

1. In the Integration Server Administrator, click **Settings > Extended**.
2. Click **Show and Hide Keys**.

Look for the `watt.server.auth.skipForMediator` property and ensure it is set to true.

3. If the `watt.server.auth.skipForMediator` property is not present, add it as follows:
 - a. Click **Edit Extended Settings**.
 - b. Type `watt.server.auth.skipForMediator=true` on a separate line.
 - c. Click **Save**.
 - d. Restart Integration Server.

Effective Policies

When you publish an API to Mediator, CentraSite automatically validates the API's policy enforcement workflow to ensure that:

CentraSite will inform you of any violation, and you will need to correct the violations before publishing the API.

When you publish an API to Mediator, CentraSite combines the actions specified within the proxy API's enforcement definition, and generates what is called the effective policy for the API. For example, suppose your API is configured with two run-time actions: one that performs a logging action and another that performs a security action. When you publish the API, CentraSite automatically combines the two actions into one effective policy. The effective policy, which contains both the logging action and the security action, is the policy that CentraSite actually publishes to Mediator with the API.

When CentraSite generates the effective policy, it validates the resulting action list to ensure that:

- Any action that appears in a single message flow multiple times is allowed to appear multiple times.

For those actions that can appear in a message flow only once (for example, Evaluate IP Address), Mediator will choose only one, which might cause problems or unintended results.

- All action dependencies are properly met. That is, some actions must be used in conjunction with another particular action.

If the list contains conflicts or inconsistencies, CentraSite resolves them according to Policy Resolution Rules.

The effective policy that CentraSite produces for an API is contained in an object called a virtual service definition (VSD). The VSD is given to Mediator when you publish the API. After you publish an API to Mediator, you can view its VSD (and thus examine the effective policy that CentraSite generated for it) from the Mediator user interface.

The following table shows:

- Action is WS-Security Policy 1.2 compliant.
- Action dependencies, that is, whether an action must be used in conjunction with another particular action.
- Action exclusives, that is, whether an action cannot be used in conjunction with another particular action.
- Action occurrences, that is, whether an action can occur once or multiple times within a message flow stage. An action can occur multiple times in a policy if the selection criteria is combined using an AND operator (not an OR operator).

Action		
Require HTTP / HTTPS	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	Require JMS
	Once or multiple in a policy?	Once
Require JMS	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	Require HTTP / HTTPS
	Once or multiple in a policy?	Once
Request Transformation	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Multiple
Invoke webMethods Integration Server	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Multiple

Action		
Enable REST Support	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	WS-Security based actions
	Once or multiple in a policy?	Once
Set Media Type	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Require SSL	WS-Security Policy Compliant	Yes
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Require WSS SAML Token	WS-Security Policy Compliant	Yes
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once

Action		
Require Signing	WS-Security Policy Compliant	Yes
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Require Encryption	WS-Security Policy Compliant	Yes
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Require Timestamps	WS-Security Policy Compliant	Yes
	Dependency Requirement	At least one of the following actions: <ul style="list-style-type: none"> ■ Evaluate WSS Username Token ■ Evaluate WSS X.509 Certificate ■ Require Signing ■ Require Encryption
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Evaluate Kerberos Token	WS-Security Policy Compliant	Yes

Action		
	Dependency Requirement	None
	Mutually Exclusive	No
	Once or multiple in a policy?	Once
Evaluate OAuth2 Token	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Evaluate HTTP Basic Authentication	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	<ul style="list-style-type: none"> ■ Evaluate OAuth2 Authentication ■ OAuth2 Authentication ■ NTLM Authentication
	Once or multiple in a policy?	Once
Evaluate WSS Username Token	WS-Security Policy Compliant	Yes
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once

Action		
Evaluate WSS X.509 Certificate	WS-Security Policy Compliant	Yes
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Evaluate IP Address	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Evaluate XPath Expression	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Evaluate Hostname	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once

Action		
Evaluate Client Certificate for SSL Connectivity	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Log Invocation	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Monitor Service Level Agreement	WS-Security Policy Compliant	No
	Dependency Requirement	At least one Evaluate action, or the Require WSS SAML Token.
	Mutually Exclusive	None
	Once or multiple in a policy?	Multiple
Monitor Service Performance	WS-Security Policy Compliant	No
	Dependency Requirement	At least one Evaluate action, or the Require WSS SAML Token.
	Mutually Exclusive	None

Action		
	Once or multiple in a policy?	Multiple
Throttling Traffic Optimization	WS-Security Policy Compliant	No
	Dependency Requirement	At least one of the "Evaluate" actions, or the Require WSS SAML Token, provided the Alert for Consumer Applications value is specified.
	Mutually Exclusive	None
	Once or multiple in a policy?	Multiple
Validate Schema	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
HTTP Basic Authentication	WS-Security Policy Compliant	No
	Dependency Requirement	At least one Routing based action.
	Mutually Exclusive	<ul style="list-style-type: none"> ■ NTLM Authentication ■ OAuth2 Authentication ■ JMS Routing Rule ■ Evaluate OAuth2 Authentication

Action		
	Once or multiple in a policy?	Once
NTLM Authentication	WS-Security Policy Compliant	No
	Dependency Requirement	At least one Routing based action.
	Mutually Exclusive	<ul style="list-style-type: none"> ■ HTTP Basic Authentication ■ OAuth2 Authentication ■ JMS Routing Rule ■ Evaluate HTTP Basic Authentication ■ Evaluate OAuth2 Authentication
	Once or multiple in a policy?	Once
OAuth2 Authentication	WS-Security Policy Compliant	No
	Dependency Requirement	At least one of the "Routing" actions.
	Mutually Exclusive	<ul style="list-style-type: none"> ■ HTTP Basic Authentication ■ NTLM Authentication ■ JMS Routing Rule ■ Evaluate HTTP Basic Authentication
	Once or multiple in a policy?	Once
JMS Routing Rule	WS-Security Policy Compliant	No

Action		
	Dependency Requirement	None
	Mutually Exclusive	Routing based actions
	Once or multiple in a policy?	Once
Set Message Properties	WS-Security Policy Compliant	No
	Dependency Requirement	JMS Routing Rule
	Mutually Exclusive	Routing based actions
	Once or multiple in a policy?	Once
Set JMS Headers	WS-Security Policy Compliant	No
	Dependency Requirement	JMS Routing Rule
	Mutually Exclusive	Routing based actions
	Once or multiple in a policy?	Once
Straight Through Routing	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	Routing based actions
	Once or multiple in a policy?	Once
Content Based Routing	WS-Security Policy Compliant	No

Action		
	Dependency Requirement	None
	Mutually Exclusive	Routing based actions
	Once or multiple in a policy?	Once
Load Balancing and Failover Routing	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	Routing based actions
	Once or multiple in a policy?	Once
Context Based Routing	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	Routing based actions
	Once or multiple in a policy?	Once
Set Custom Headers	WS-Security Policy Compliant	No
	Dependency Requirement	At least one Routing based action.
	Mutually Exclusive	None
	Once or multiple in a policy?	Once
Response Transformation	WS-Security Policy Compliant	No

Action		
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Multiple
Invoke webMethods Integration Server	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Multiple
Conditional Error Processing	WS-Security Policy Compliant	No
	Dependency Requirement	None
	Mutually Exclusive	None
	Once or multiple in a policy?	Once

Usage Cases for Identifying/Authenticating Clients

When deciding which type of identifier to use to identify a client, consider the following points:

- Whatever identifier you choose to identify a client, it must be unique to the application. Identifiers that represent user names are often not suitable because the identified users might submit requests for multiple APIs.
- Identifying applications by IP address or host name is often a suitable choice, however, it does create a dependency on the network infrastructure. If a client moves to a new machine, or its IP address changes, you must update the identifiers in the application asset.

- Using X.509 certificates or a custom token that is extracted from the SOAP message itself (using an XPATH expression), is often the most trouble-free way to identify a client.

Following are some common combinations of actions used to authenticate/identify clients.

- **Scenario 1: Identify clients by IP address or host name**

- The simplest way to identify clients is to use the [Evaluate IP Address](#) action.

- **Scenario 2: Authenticate clients by HTTP authentication token**

Use the following actions:

- [Evaluate HTTP Basic Authentication](#) to identify clients using the token derived from the HTTP Header.
- [HTTP Basic Authentication](#).

- **Scenario 3: Authenticate clients by WS-Security authentication token**

Use the following action:

- [Evaluate WSS Username Token](#) action to identify clients using the token derived from the WSS Header.

- **Scenario 4: Authenticate clients by WSS X.509 certificate**

- [Evaluate WSS X.509 Certificate](#) action to identify clients using the WSS X.509 certificate.
- [Require SSL](#) action.

Run-Time Actions Reference

This section provides an alphabetic list of the built-in run-time actions you can include in the run-time governance rules for APIs.

Allow Anonymous Usage

When an API is not configured with any of the **Security** policy actions, Mediator routes the incoming requests directly to the native API, irrespective of the client credentials passed in the request.

When an API is configured with a **Security** policy action, Mediator routes the incoming requests to the native API that are successfully identified using the client credentials passed in the request. When this API is also configured with the Allow Anonymous Usage policy action, Mediator routes all of the incoming requests to the native API. However, the successfully identified requests are grouped under the respective identified consumer, and all unidentified requests are grouped under a common consumer named as unknown.

With this action, you can perform the following list of consumer-specific actions, while also allowing the requests to pass through to the native API:

- View the runtime events for a particular consumer.
- Monitor the Service Level Agreement for a few consumers and send an alert email based on specific criteria's, for example, request count, availability, and so on.
- Throttle requests from a particular consumer, and restrict requests from that consumer if the number of requests to the API reaches the configured hard limit during a specified period of time.

Input Parameters

Allow Anonymous Usage (Boolean). Specifies whether to allow all the incoming requests to access the API, without restriction.

Value	Description
True	(Default). Allows all of the incoming requests to access an API even if the request is not identified using the Security policies that are configured for that API.
False	Allows only the incoming requests that are identified using the configured Security policies to access the API.

Content Based Routing

If you have a native API that is hosted at two or more endpoints, you can use the Content Based Routing to route specific types of messages to specific endpoints.

You can route messages to different endpoints based on specific values that appear in the request message.

When this action is configured for a proxy API, the requests are routed according to the routing rules you create. That is, they are routed based on the successful evaluation of one or more XPath expressions that are constructed utilizing the content of the request payload. For example, a routing rule might allow requests for half of the methods of a particular service to be routed to Endpoint A, and the remaining methods to be routed to Endpoint B.

Input Parameters

Route To *URI. Mandatory.* Enter the URL of the native API endpoint to route the request to in case all routing rules evaluate to False. For example:

```
http://mycontainer/creditCheckService
```

Click the **Configure Endpoint Properties** icon (next to the **Route To** field) if you want to configure a set of properties for the specified endpoint.

Alternatively, Mediator offers "Local Optimization" capability if the native endpoint is hosted on the same Integration Server as Mediator. With local optimization, API invocation happens in-memory and not through a network hop.

Specify the native API in either of the following forms:

```
local://<Service-full-path>
```

OR

```
local://<server>:<port>/ws/<Service-full-path>
```

For example:

```
local://MyAPIFolder:MyLocalAPI
```

which points to the endpoint API `MyLocalAPI` which is present under the folder `MyAPIFolder` in Integration Server.

Note: Local Optimization is not applicable to REST based APIs.

Add Routing Rule (button)

Click the **Add Routing Rule** button and complete the **Add Routing Rule** dialog box as follows.

1. In the **XPath Expression** field, specify an argument to evaluate the XPath expression contained in the request.
2. To add a custom **Namespace**, specify a name and value for the namespace in the **Prefix** and **URI** fields. If you need to add additional rows, use the plus button.
3. In the **Route To** field, specify the URL of the native API to route the request to, if the rule criteria are met.
4. Click the **Configure Endpoint Properties** icon (next to the **Route To** field) if you want to configure a set of properties for the specified endpoint individually.
5. Click **OK**.

Configure Endpoint Properties (icon)



Optional. This icon displays the **Endpoint Properties** dialog box that enables you to configure a set of properties for the Mediator to route incoming requests to the native API as follows:

SOAP
Optimization
Method

Only for SOAP-Based APIs. Mediator can use the following optimization methods to parse SOAP requests to the native API:

<u>Value</u>	<u>Description</u>
MTOM	Mediator will use the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API.
SwA	Mediator will use the SOAP with Attachment (SwA) technique to parse SOAP requests to the API.
None	<i>Default.</i> Mediator will not use any optimization method to parse the SOAP requests to the API.

Note:Keep the following points in mind:

- Bridging between SwA and MTOM is not supported. If a client sends a SwA request, Mediator can only forward SwA to the native API. The same is true for MTOM, and applies to responses received from the native API. That is, a SwA or MTOM response received by Mediator from a native API will be forwarded to the client using the same format it received.
- When sending SOAP requests that do not contain a MTOM or SWA attachment to a native API that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related'. This is necessary so Mediator knows how to parse the response properly.

HTTP
Connection
Timeout

Number.Optional. The time interval (in seconds) after which a connection attempt will timeout. If a value 0 is specified (or if the value is not specified), Mediator will use the value specified in the `Connection Timeout` field

(in the Integration Server Administrator, go to **Settings > Extended**). Default: 30 seconds.

Read Timeout *Number.Optional.* The time interval (in seconds) after which a socket read attempt will timeout. If a value 0 is specified (or if the value is not specified), Mediator will use the value specified in the `Read Timeout` field (in the Integration Server Administrator, go to **Settings > Extended**). Default: 30 seconds.

SSL Configuration *Optional.* To enable SSL client authentication that Mediator will use to authenticate incoming requests for the native API, you must specify values for both the `Client Certificate Alias` field and the `IS Keystore Alias` field. If you specify a value for only one of these fields, a deployment error will occur.

Note: SSL client authentication is optional; you may leave both fields blank.

Prerequisite: You must set up the key alias and keystore properties in the Integration Server. For the procedure, see *webMethods Integration Server Administrator's Guide*.

You will use these properties to specify the following fields:

Value	Description
<code>Client Certificate Alias</code>	<i>Mandatory.</i> The client's private key to be used for performing SSL client authentication.
<code>IS Keystore Alias</code>	<i>Mandatory.</i> The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of <code>Client Certificate Alias</code>) will be used for performing SSL client authentication.

WS Security Header *Only for SOAP-Based APIs.* Indicates whether Mediator should pass the WS-Security

headers of the incoming requests to the native API.

<u>Value</u>	<u>Description</u>
Remove processed security headers	Removes the security header if it is processed by Mediator (that is, if Mediator processes the header according to the API's security run-time action). Note that Mediator will not remove the security header if both of the following conditions are true: 1) Mediator did not process the security header, and 2) the <code>mustUnderstand</code> attribute of the security header is <code>0/false</code>).
Pass all security headers	<i>Default.</i> Passes the security header, even if it is processed by Mediator (that is, even if Mediator processes the header according to the API's security action).

Context Based Routing

If you have a native API that is hosted at two or more endpoints, you can use the Context Based Routing to route specific types of messages to specific endpoints.

When this action is configured for a proxy API, the requests are routed according to the routing rules you create. A routing rule specifies where requests should be routed, and the criteria by which they should be routed there. For example, requests can be routed according to certain clients, certain dates/times, or according to requests that exceed/fall below a specified metric (Total Count, Success Count, Fault Count, and so on). You can create one or more rules.

Input Parameters

Default Route To *URI. Mandatory.* Enter the URL of the native API endpoint to route the request to in case all routing rules evaluate to False. For example:

`http://mycontainer/creditCheckService`

Click the **Configure Endpoint Properties** icon (next to the **Route To** field) if you want to configure a set of properties for the specified endpoint.

Alternatively, Mediator offers "Local Optimization" capability if the native endpoint is hosted on the same Integration Server as Mediator. With local optimization, API invocation happens in-memory and not through a network hop.

Specify the native API in either of the following forms:

```
local://<Service-full-path>
```

OR

```
local://<server>:<port>/ws/<Service-full-path>
```

For example:

```
local://MyAPIFolder:MyLocalAPI
```

which points to the endpoint API `MyLocalAPI` which is present under the folder `MyAPIFolder` in Integration Server.

Note: Local Optimization is not applicable to REST based APIs.

Add Routing Rule (button)

Click the **Add Routing Rule** button and complete the **Add Routing Rule** dialog box as follows.

1. In the **Name** field, specify a name for the routing rule.
2. In the **Condition** panel, specify the following as required:
 - a. In the **Variable** column, select **Time**, **IP Address Range**, **Date**, **Consumer**, **Predefined Context Variable** or **Custom Context Variable**.
 - b. In the **Value** column, specify an applicable value. For **Date** choose **Before**, **After** or **Equal To** and enter a date. For **Time** choose **Before** or **After** and enter a time. For **IP Address**, enter numeric values for **Between** and **And**. For **Consumer**, enter a consumer application name in the text box. For **Predefined Context Variable** or **Custom Context Variable**, choose the **String** or **Integer** data type. Select a predefined variable name or custom variable name from the drop-down list. For **String**, choose **Equal To** or **Not Equal To** and enter a value. For **Integer**, choose **Greater Than**, **Less Than**, **Not Equal To**, **Equal To** or and enter a value.

Note: Keep the following points in mind:

- For the list of the predefined context variables, see *Using Context Variables in APIs*.
- The predefined context variable `PROTOCOL_HEADER` is not available in the drop-down list; to include `PROTOCOL_HEADER` in the rule, define the variable as Custom Context Variable.

- If you define a custom context variable in the routing rule, you must write a *webMethods IS service* and invoke it in the API's Context Based Routing action. In this Integration Server service, use the API to get/set the custom context variable.

For more information, see "[The API for Context Variables](#)" on page 279

If you need to specify multiple variables, use the plus button to add rows.

- If you have more than one routing rule, choose an operator for the expression: **AND** or **OR** (the default).
- In the **Route To** field, specify the URL of the native API endpoint to route the request to, if the rule criteria are met.
- Click the **Configure Endpoint Properties** icon (next to the **Route To** field) if you want to configure a set of properties for the specified endpoint individually.
- Click **OK**.

Configure Endpoint Properties (icon)

Optional. This icon displays the **Endpoint Properties** dialog box that enables you to configure a set of properties for the Mediator to route incoming requests to the native API as follows:

SOAP
Optimization
Method

Only for SOAP-Based APIs. Mediator can use the following optimization methods to parse SOAP requests to the native API:

<u>Value</u>	<u>Description</u>
MTOM	Mediator will use the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API.
SwA	Mediator will use the SOAP with Attachment (SwA) technique to parse SOAP requests to the API.
None	<i>Default.</i> Mediator will not use any optimization method to parse the SOAP requests to the API.

Note: Keep the following points in mind:

- Bridging between SwA and MTOM is not supported. If a client sends a SwA request, Mediator can only forward SwA to the native API. The same is true for MTOM, and applies to responses received from the native API. That is, a SwA or MTOM response received by Mediator from a native API will be forwarded to the client using the same format it received.
- When sending SOAP requests that do not contain a MTOM or SWA attachment to a native API that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related'. This is necessary so Mediator knows how to parse the response properly.

HTTP
Connection
Timeout

Number.Optional. The time interval (in seconds) after which a connection attempt will timeout. If a value 0 is specified (or if the value is not specified), Mediator will use the value specified in the `Connection Timeout` field (in the Integration Server Administrator, go to **Settings > Extended**). Default: 30 seconds.

Read
Timeout

Number.Optional. The time interval (in seconds) after which a socket read attempt will timeout. If a value 0 is specified (or if the value is not specified), Mediator will use the value specified in the `Read Timeout` field (in the Integration Server Administrator, go to **> Settings > Extended**). Default: 30 seconds.

SSL
Configuration

Optional. To enable SSL client authentication that Mediator will use to authenticate incoming requests for the native API, you must specify values for both the Client Certificate Alias field and the IS Keystore Alias field. If you specify a value for only one of these fields, a deployment error will occur.

Note: SSL client authentication is optional; you may leave both fields blank.

Prerequisite: You must set up the key alias and keystore properties in the Integration Server. For the procedure, see *webMethods Integration Server Administrator's Guide*.

You will use these properties to specify the following fields:

	<u>Value</u>	<u>Description</u>
	Client Certificate Alias	<i>Mandatory.</i> The client's private key to be used for performing SSL client authentication.
	IS Keystore Alias	<i>Mandatory.</i> The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) will be used for performing SSL client authentication.
WS Security Header		Only for SOAP-Based APIs. Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native API.
	<u>Value</u>	<u>Description</u>
	Remove processed security headers	<i>Default.</i> Removes the security header if it is processed by Mediator (that is, if Mediator processes the header according to the API's security runtime action). Note that Mediator will not remove the security header if both of the following conditions are true: 1) Mediator did not process the security header, and 2) the mustUnderstand attribute of the security header is 0/false).
	Pass all security headers	Passes the security header, even if it is processed by Mediator (that is, even if Mediator processes the header according to the API's security action).

Conditional Error Processing

This action returns a custom error response (and/or the native provider's service fault content) to the client when the native provider returns a service fault. You can configure conditional error processing and use variables to create custom error messages. This action also allows you to define error messages based on content types: text, XML, and JSON.

Alternatively, you can configure global error responses for all APIs, using Mediator's Service Fault Configuration page (see *Administering webMethods Mediator*).

Note: To enable support for conditional error processing in CentraSite, you must add an entry in the custom \conf\centrasite.xml file as follows: `<Action name="ConditionalErrorProcessing" id="uddi:61849291-3bad-4612-819b-5fe5a6b4e958" occurrence="*"></Action>`. For details on configuring the centrasite.xml file, see CentraSite documentation.

Input Parameters

Error Conditions Specifies the error conditions and how each of these error conditions are to be processed. You can define a maximum of three conditions. You can configure only one error condition per type. For example, you cannot define two status code conditions. To specify multiple conditions, use the plus button to add rows.

Type	Value	Description
	Status Code	<i>Default</i> Specify the error status code. For example: 400, 500.
	HTTP Header	Specify the details of the custom HTTP header(s) included in the client requests. The parameters are: <ul style="list-style-type: none"> ■ HTTP Name<i>String</i>. Specifies the name of the HTTP header. ■ HTTP Value<i>String</i>. Specifies the value of the HTTP header.
	XPath Expression	Specifies the details of the XPath expression in the API request. The parameters are: <ul style="list-style-type: none"> ■ XPath Expression ■ Namespaces: Specifies the namespace of the XPath expression. To specify multiple XPath expressions, use the plus button to add rows. <ul style="list-style-type: none"> ■ Prefix: The prefix for the namespace. For example, soapenv or axis ■ URI: The namespace URI - For example, http://schemas.xmlsoap.org/

soap/envelope/ or http://
ws.apache.org/axis

- Value: Specifies the value of the XPath expression.

Pre-
Processing

Specifies how the native service error response is to be processed before sending the same to Mediator. You can configure multiple ESB and XSLT processing steps. You can either log the error message sent by the native service without any changes or you can remove any critical information that you do not want Mediator to log or send to the client. The processing of these steps is taken up in the order in which they are configured.

Value	Description
XSLT	<i>String.Optional.</i> Specify the XSLT file that you want to use to transform the service error response. Use the Choose button to browse to and select a file.
ESB	<i>Default. String.Optional.</i> Specify the webMethods IS Service . Invokes one or more webMethods IS services to manipulate the response message from the native API before it is returned to the consuming application. The IS service will have access to the response message context (the axis2 MessageContext instance) before it is updated with the custom error message. For example, you might want to send emails or perform custom alerts based on the response payload.

Failure
Message

String. Specify the custom failure message that Mediator should send to the client. You can configure an error template each for the content types: Text, XML, and JSON. For addition details, see the fault handler variables listed in the table below. For example, \$CONSUMER_APPLICATION \$OPERATION :\$ERROR_MESSAGE.

Value	Description
Content-Type	Specifies the content type for which the failure message is defined. You can select the following content types:

- text
- xml
- json

NoteYou can define only one error template per content type.

Error Template	Specifies the content of the error template to use. You can use predefined fault handler variables to create the error template. A list of the predefined variables is available in the table below.
Use as Default	Specifies the error template to use as the default.
Custom Error Variables	Specify the error variables to be used in the custom error message. To specify multiple error variables, use the plus button to add rows.
Payload Type	Select the payload type.
Value	Description
Request	Selects the request payload type.
Response	Selects the response payload type.
Name	Specifies the name of the payload.
XPath Expression	Specifies the details of the XPath expression in the API request.
Namespaces	Specifies the namespace of the XPath expression. To specify multiple namespaces , use the plus button to add rows.
Prefix	The prefix for the namespace.
URI	The namespace URI.
Post-Processing	You can configure multiple ESB and XSLT processing steps. You can either log the error message sent by the native service without any changes or you can remove any critical information that you do not

want Mediator to log or send to the client. The processing of these steps is taken up in the order in which they are configured.

Value	Description
XSLT	<i>String.Optional.</i> Specify the XSLT file that you want to use to transform the service error response. Use the Choose button to browse to and select a file.
ESB	<i>Default. String.Optional.</i> Specify the webMethods IS Service . Invokes one or more webMethods IS services to manipulate the API fault after the Conditional Error Processing action is invoked. The IS service will have access to the entire API fault and the Conditional Error Processing message. You can make further changes to the fault message structure, if needed.

Send Native Provider Fault Message	When the parameter is enabled, Mediator sends the native SOAP / REST failure message to the client. When you enable this parameter, the <code>Failure Message</code> is ignored when a fault is returned by the native API provider. (Faults returned by internal Mediator exceptions will still be handled by the <code>Failure Message</code> .)
------------------------------------	--

Failure Messages

The failure message is returned in both of the following cases:

- When a failure is returned by the native API provider.
In this case, the `$ERROR_MESSAGE` variable in the failure message will contain the message produced by the provider's exception that caused the error. This is equivalent to the `getMessage` call on the Java Exception.
- When a failure is returned by internal Mediator exceptions (such as policy violation errors, timeouts, and so on).

In this case, `$ERROR_MESSAGE` will contain the error message generated by Mediator.

Alternatively, you can configure global failure messages for *all* APIs, using Mediator's `Service Fault Configuration` page, as described in *Administering webMethods Mediator*.

Mediator returns the following failure message to the consuming application:

```
Mediator encountered an error:$ERROR_MESSAGE while executing
operation:$OPERATION service:$SERVICE at time:$TIME on date:$DATE.
```

```
The client ip was:$CLIENT_IP. The current user:$USER.
The consumer application:$CONSUMER_APPLICATION".
```

The precedence of the Failure Message configurations is as follows:

- If you configure a Conditional Error Processing action for an API, the failure message configurations take precedence over any settings on the global Service Fault Configuration page.
- If you do not configure a Conditional Error Processing action for an API, the settings on the Service Fault Configuration page take precedence.

The default failure message contains predefined fault handler variables (\$ERROR_MESSAGE, \$OPERATION, and so on.) which are described in the table below.

You can customize the default failure message using the following substitution variables, where Mediator replaces the variable reference with the real content at run time:

- The predefined context variables listed in "[The Predefined Context Variables](#)" on page 274.
- Custom context variables that you declare using Mediator's API (see "[The API for Context Variables](#)" on page 279).

Note: If you want to reference a custom context variable that you have already defined in a virtual API (as opposed to one you have [The API for Context Variables](#)), you must add the prefix \$mx to the variable name in order to reference the variable. For example, if you defined the variable TAXID, you would reference it as \$mx:TAXID.

The fault handler variables are described below.

Note: If no value is defined for a fault handler variable, then the returned value will be the literal string "null". For example, \$CONSUMER_APPLICATION will always be "null" if the service's policy does not contain at least one of the "Evaluate" actions.

Fault Handler Variable	Description
\$ERROR_MESSAGE	The error message produced by the exception that is causing the error. This is equivalent to the getMessage call on the Java Exception. This maps to the faultString element for SOAP 1.1 or the Reason element for SOAP 1.2 catch.
\$OPERATION	The operation that was invoked when this error occurred.
\$SERVICE	The service that was invoked when this error occurred.

Fault Handler Variable	Description
\$TIME	The time (as determined on the Container side) at which the error occurred.
\$DATE	The date (as determined on the Container side) at which the error occurred.
\$CLIENT_IP	The IP address of the client invoking the service. This might be available for only certain invoking protocols, such as HTTP(S).
\$USER	The currently authenticated user. The user will be present only if the Transport/SOAP Message have user credentials.
\$CONSUMER_APPLICATION	The currently identified consumer application (client).
\$NATIVE_STATUS_CODE	The HTTP error status code that is returned by the native service.

Enable REST Support

This action enables REST support for an existing SOAP based API. You would need to configure the **Enable REST Support** action to:

- Expose a SOAP API both as a SOAP based API and a REST based API using Mediator. Clients who can only send REST requests can now invoke a REST-enabled SOAP API using both a SOAP request and a REST request in Mediator.
- Expose a SOAP API as a REST based API using API-Portal. Clients can now invoke a REST-enabled SOAP API using a REST request in API-Portal.

Important: This action is applicable only for a SOAP API.

Note: This action is set by default in the **Receive** step for all SOAP APIs. To disable the REST support for a SOAP API, delete the Enable REST Support action in the **Receive** step for the API.

Input Parameters

None.

Evaluate Client Certificate for SSL Connectivity

If you have a native API that requires to authenticate a client to the Integration Server using the Secure Sockets Layer (SSL) client authentication, you can use the Evaluate

Client Certificate action to extract the client's identity certificate, and verify the client's identity (certificate-based authentication).

This form of authentication does not occur at the message layer using a user ID and password or tokens. This authentication occurs during the connection handshake using SSL certificates.

This action extracts the client identity certificate supplied by the client to the Mediator during the SSL handshake over the Transport layer. For example, when you have configured this action for a proxy API, the PEP extracts the certificate from the Transport layer. In order to identify clients by transport-level certificates, the run-time communication between the client and the Mediator must be over HTTPS and the client must pass a valid certificate.

To use this action, the following prerequisites must be met:

- In Integration Server, create a keystore and truststore, as described in *webMethods Integration Server Administrator's Guide*.
- In Integration Server, create an HTTPS port as described in the *webMethods Integration Server Administrator's Guide*.
- Configure Mediator by setting the HTTPS Ports Configuration parameter, as described in *Administering webMethods Mediator*.

Mediator rejects requests that do not include a client certificate during the SSL handshake over the Transport layer.

If Mediator cannot identify the client, Mediator fails the request and generates a Policy Violation event.

Input Parameters

Identify Consumer *String*. The list of consumers against which the client certificate should be validated for identifying requests from a particular client.

<u>Value</u>	<u>Description</u>
Registered Consumers	Mediator will try to verify the client identify certificate against the list of consumer applications who are registered as consumers for the specified API.
Global Consumers	<i>Default</i> . Mediator will try to verify the client identify certificate against a list of all global consumers available in the Mediator.

Evaluate Hostname

If you have a native API that requires to authenticate a client to the Integration Server using the hostname, you can use the Evaluate Hostname action to extract the client's hostname from the HTTP request header, and verify the client's identity.

This action extracts the specified hostname from an incoming request and locates the client defined by that hostname. For example, when you have configured this action for an API, the PEP extracts the hostname from the request's HTTP header at run time and searches its list of consumers for the client defined by the hostname.

Mediator will evaluate the incoming request to identify and validate that the client's request originated from a particular host machine.

Mediator rejects requests that do not include the hostname of an Integration Server user.

If Mediator cannot identify the client, Mediator fails the request and generates a Policy Violation event.

Input Parameters

Identify Consumer *String.* The list of consumers against which the hostname should be validated for identifying requests from a particular client.

<u>Value</u>	<u>Description</u>
Registered Consumers	Mediator will try to verify the client's hostname against the list of consumer applications who are registered as consumers for the specified API.
Global Consumers	<i>Default.</i> Mediator will try to verify the client's hostname against a list of all global consumers available in the Mediator.

Evaluate HTTP Basic Authentication

If you have a native API that requires to authenticate a client to the Integration Server using the HTTP Basic Authentication, you can use the Evaluate HTTP Basic Authentication action to extract the client's credentials (user ID and password) from the Authorization request header, and verify the client's identity.

This action uses HTTP Basic authentication to verify the client's authentication credentials contained in the request's Authorization header. When this action is configured for an API, Mediator validates the credentials against the list of consumers available in the Integration Server on which Mediator is running. If you have chosen the checkbox `Authenticate User` using the HTTP Basic Authentication, this type of client authentication is referred to as "preemptive authentication".

If the user/password value in the Authorization header cannot be authenticated as a valid Integration Server user (or if the Authorization header is not present in the request), a 500 SOAP fault is returned, and the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. This type of client authentication is referred to as “non-preemptive authentication”. If the client does not successfully respond to the challenge, a 401 “WWW-Authenticate: Basic” response is returned and the invocation is not routed to the policy engine.

If you choose to omit the `Authenticate User` parameter (and regardless of whether an Authorization header is present in the request or not), then Mediator forwards the request to the native API, without attempting to authenticate the request.

In the case where a client sends a request with transport credentials (HTTP Basic Authentication) and message credentials (WSS Username Token or WSS X.509 Token), the message credentials take precedence over the transport credentials when Integration Server determines which credentials it should use for the session. For more information, see ["Evaluate WSS Username Token" on page 424](#) and ["Evaluate WSS X.509 Certificate" on page 426](#).

If Mediator cannot identify the client, Mediator fails the request and generates a Policy Violation event.

Input Parameters

`Identify Consumer` *String*. The list of consumers against which authentication credentials (user ID and password) should be validated for identifying requests from a particular client.

<u>Value</u>	<u>Description</u>
Registered Consumers	Mediator will try to verify the client's credentials against the list of consumer applications who are registered as consumers for the specified API.
Global Consumers	<i>Default.</i> Mediator will try to verify the client's credentials against a list of all global consumers available in the Mediator.
Do Not Identify	Mediator forwards the request to the native API, without attempting to verify client's credentials in incoming request.

`Authenticate User` Use this checkbox to specify the users who can access the APIs. If you select the checkbox, Mediator allows only the users specified in the `Identify Consumer` parameter to access the APIs. If you do not select the checkbox, Mediator allows all users to access

the API. In this case, do not configure the `Identify Consumer` parameter.

Note: If you have selected the `Authenticate User` option, the client that connects to the API must have an `Integration Server` user account.

Evaluate IP Address

If you have a native API that requires to authenticate a client to the `Integration Server` using the IP address, you can use the `Evaluate IP Address` action to extract the client's IP address from the HTTP request header, and verify the client's identity.

This action extracts the specified IP address from an incoming request and locates the client defined by that IP address. For example, when you have configured this action for a proxy API, the PEP extracts the IP address from the request's HTTP header at run time and searches its list of consumers for the client defined by the IP address.

Mediator will evaluate the incoming request to identify and validate that the client's request originated from a particular IP address.

Mediator rejects requests that do not include the IP address of an `Integration Server` user.

If Mediator cannot identify the client, Mediator fails the request and generates a `Policy Violation` event.

Input Parameters

`Identify Consumer`

String. The list of consumers against which the IP address should be validated for identifying requests from a particular client.

Value	Description
<code>Registered Consumers</code>	Mediator will try to verify the client's credentials against the list of consumer applications who are registered as consumers for the specified API. Mediator will evaluate whether the request header contains the <code>X-Forwarded-For</code> , which is used for identifying the IP address of a client through an HTTP proxy.
<code>Global Consumers</code>	<i>Default.</i> Mediator will try to verify the client's credentials against a list of

all global consumers available in the Mediator.

Do Not
Identify

Mediator forwards the request to the native API, without attempting to verify client's credentials in incoming request.

Evaluate Kerberos Token

Evaluate Kerberos Token policy can be used in any of the following scenarios:

- when the native service does not support Kerberos authentication.
- when you want to centrally configure Kerberos authentication in Mediator for services where Mediator is configured to forward the request to a clustered group of native servers through load balancer.

Note: For Evaluate Kerberos Token policy, JMS and HTTP are not supported as inbound protocols. Also, you must select HTTPS as the inbound protocol to enforce this policy. Evaluate Kerberos Token policy complies to the **KerberosOverTransport** section described in the following article, <https://msdn.microsoft.com/en-us/library/aa751836.aspx>. Kerberos inbound authentication support is available at message level and not at transport level.

Input Parameters

Service Principal Name *String.Mandatory.* A valid Service Principal Name (SPN). The specified value will be used by the client to obtain a service ticket from the KDC server. The SPN is created in the Active Directory (AD) by the AD domain administrator using the following command:

```
setspn -a <domain name>\<username> spnname
```

For example,

```
setspn -a eur\user1 spnname
```

Note: Service Principal Name is currently only supported as a user name based form and not a service name based form.

Service Principal Password *String.Mandatory.* A valid password of the SPN user. For example, if the setspn command is set for the domain user *eur\user1*, this field represents the password set for the domain user *eur\user1*.

Identify Consumer *String*. The list of consumers against which the Kerberos token must be validated for identifying requests from a particular client.

Value	Description
Do Not Identify	Mediator forwards the request to the native API, without identifying the consumer application(in global/registered consumer list) that corresponds to the principal identified after successful Kerberos authentication.
Registered Consumers	Mediator will try to identify the consumer based on principal that it set after successful Kerberos authentication against the list of consumer applications who are registered as consumers for the specified API.
Global Consumers	<i>Default</i> . Mediator will try to identify the consumer based on principal that it set after successful Kerberos authentication against the list of global consumer applications in Mediator.

Evaluate OAuth2 Token

If you have a native API that requires to authenticate a client to the Integration Server using the OAuth 2.0 credentials (access token), you can use the Evaluate OAuth2 Authentication action to extract the client's credentials from the HTTP request header, and verify the client's identity.

This action extracts the specified OAuth access token from an incoming request and locates the client defined by that access token. For example, when you have configured this action for an API, the PEP extracts the OAuth access token from the request's HTTP header at run time and searches its list of consumers for the client that is defined by that access token.

Mediator will evaluate the incoming request to identify and validate that the client's access token.

Mediator rejects requests that do not include the OAuth access token of an Integration Server user.

Mediator supports OAuth2.0 using the grant type "Client Credentials".

If Mediator cannot identify the client, Mediator fails the request and generates a Policy Violation event.

Input Parameters

Identify User *String*. The list of consumers against which the OAuth token should be validated for identifying requests from a particular client.

<u>Value</u>	<u>Description</u>
Registered Consumers	Mediator will try to verify the client's OAuth access token against the list of consumer applications who are registered as consumers for the specified API.
Global Consumers	<i>Default</i> . Mediator will try to verify the client's OAuth access token against a list of all global consumers available in the Mediator.

Validate Access Token *Boolean.Optional*. This option uses your resource server to verify clients. When Integration Server acts as a resource server, it receives requests from clients that include an access token. The resource server asks the authorization server to validate the access token and user. If the token is valid and the user has privileges to access the folders and services, the resource server executes the request.

For more information about using Integration Server to act as a resource server, see *webMethods Integration Server Administrator's Guide*.

<u>Value</u>	<u>Description</u>
True	<i>Default</i> . Mediator will verify the client's OAuth access token against the list of consumers available in the Integration Server on which Mediator is running.
False	Mediator will not verify the client's OAuth access token.

Evaluate WSS Username Token

If you have a native API that requires to authenticate a client to the Integration Server using the WS-Security authentication, you can use the Evaluate WSS Username Token action to extract the client's credentials (username token and password) from the WS-Security SOAP message header, and verify the client's identity.

This action extracts the username token and password supplied in the message header of the request and locates the client defined by that username token and password. For example, when you have configured this action for an API, the PEP extracts the username token and password from the SOAP header at run time and searches its list of consumers for the client that is defined by the credentials.

To use this action, the following prerequisites must be met:

- In Integration Server, create a keystore and truststore. For detailed information about securing communications with the server, see the *webMethods Integration Server Administrator's Guide*.
- In Integration Server, create an HTTPS port. For detailed information about configuring ports, see the *webMethods Integration Server Administrator's Guide*.
- Configure Mediator by setting the HTTPS Ports Configuration parameter. For detailed information about configuring Mediator, see *Administering webMethods Mediator*.

Mediator rejects requests that do not include the username token and password of an Integration Server user. Mediator only supports clear text passwords with this kind of authentication.

In the case where a client sends a request with transport credentials (HTTP Basic Authentication) and message credentials (WSS Username Token or WSS X.509 Certificate), the message credentials take precedence over the transport credentials when Integration Server determines which credentials it should use for the session. For more information, see ["Evaluate HTTP Basic Authentication" on page 419](#) and ["Evaluate WSS X.509 Certificate" on page 426](#).

If Mediator cannot identify the client, Mediator fails the request and generates a Policy Violation event.

Input Parameters

Identify Consumer	<i>String</i> . The list of consumers against which the username token and password should be validated for identifying requests from a particular client.
----------------------	--

Value	Description
Registered Consumers	Mediator will try to verify the client's WSS username token against the list of consumer applications who are registered as consumers for the specified API.
Global Consumers	<i>Default</i> . Mediator will try to verify the client's WSS username token against a list of all global consumers available in the Mediator.

Do Not Identify	Mediator forwards the request to the native API, without attempting to verify the client's username token in incoming request.
-----------------	--

Evaluate WSS X.509 Certificate

If you have a native API that requires to authenticate a client to the Integration Server using the WS-Security authentication, you can use the Evaluate WSS X.509 Certificate action to extract the client identity certificate from the WS-Security SOAP message header, and verify the client's identity.

This action extracts the certificate supplied in the header of an incoming SOAP request and locates the client defined by the information in that certificate. For example, when you have configured this action for an API, the PEP extracts the certificate from the SOAP header at run time and searches its list of consumers for the client that is defined by the certificate.

To use this action, the following prerequisites must be met:

- In Integration Server, create a keystore and truststore, as described in the *webMethods Integration Server Administrator's Guide*.
- In Integration Server, create an HTTPS port, as described in the *webMethods Integration Server Administrator's Guide*.
- Configure Mediator by setting the HTTPS Ports Configuration parameter, as described in *Administering webMethods Mediator*.

Mediator rejects requests that do not include the X.509 token of an Integration Server user.

In the case where a client sends a request with transport credentials (HTTP Basic Authentication) and message credentials (WSS Username Token or WSS X.509 Certificate), the message credentials take precedence over the transport credentials when Integration Server determines which credentials it should use for the session. For more information, see ["Evaluate WSS Username Token" on page 424](#) and ["Evaluate HTTP Basic Authentication" on page 419](#).

If Mediator cannot identify the client, Mediator fails the request and generates a Policy Violation event.

Input Parameters

Identify Consumer	<i>String</i> . The list of consumers against which the X.509 certificate should be validated for identifying requests from a particular client.
-------------------	--

<u>Value</u>	<u>Description</u>
--------------	--------------------

Registered Consumers	Mediator will try to verify the client's X.509 certificate against the list of consumer applications who are registered as consumers for the specified API.
Global Consumers	<i>Default.</i> Mediator will try to verify the client's X.509 certificate against a list of all global consumers available in the Mediator.
Do Not Identify	Mediator forwards the request to the native API, without attempting to verify client's certificate in incoming request.

Evaluate XPath Expression

Note: This action does not support JSON-based REST APIs.

If you have an API which includes consumer authentication using XPath, you can use the Evaluate XPath Expression action to extract the custom identification credentials from the request. You can then verify the consumer's identity using this information.

The Evaluate XPath Expression action extracts the custom authentication credentials that is supplied in the request which is represented using an XPath expression. The custom authorization can be in the form of tokens, or a username and password token combination. For example, when you configure this action for an API, the PEP extracts the custom identification from the request using an XPath expression at runtime and searches its list of consumers for the XPath defined in the global or registered consumers list.

Mediator rejects requests that do not include the XPath consumer identification defined in the global or registered consumers list.

If Mediator cannot identify the consumer, Mediator fails the request and generates a Policy Violation event.

Input Parameters

Identify Consumer	<i>String.</i> The list of consumers against which the XPath expression should be validated for identifying requests from a particular client.
-------------------	--

<u>Value</u>	<u>Description</u>
Registered Consumers	Mediator will try to verify the client's XPath expression against the list of

		consumer applications who are registered as consumers for the specified API.
	Global Consumers	<i>Default.Mediator</i> will try to verify the client's XPath expression against a list of all global consumers available in the Mediator.
	Do Not Identify	Mediator forwards the request to the native API, without attempting to verify client's XPath expression in incoming request.
Namespace	<i>String.Optional.</i> The namespace of the XPath expression to be validated.	
XPath Expression	<i>String.Mandatory.</i> An argument to evaluate the XPath expression contained in the request. See the sample below.	

Let's take a look at an example. For the following SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
</soap:Header>
  <soap:Body>
    <catalog xmlns="http://www.store.com">
      <name>My Book</name>
      <author>ABC</author>
      <price>100</price>
    </catalog>
  </soap:Body>
</soap:Envelope>
```

The XPath expression is as follows:

```
/soap:Envelope/soap:Body/catalog/author
```

To select the element or token without the namespace, use the following:

```
//*[local-name()='Envelope']/*[local-name()='Body']/*[local-name()='catalog']
/*[local-name()='author'][1]/text()
```

The result: SoftwareAG

HTTP Basic Authentication

This action uses the HTTP authentication mechanism to validate incoming requests from clients. Mediator authorizes the basic credentials (username and password) against a list of all global consumers available in the Mediator.

If the username/password value in the Authorization header cannot be authenticated as a valid Integration Server user (or if the Authorization header is not present in the request), a 500 SOAP fault is returned, and the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. If the client does not successfully respond to the challenge, a 401 "WWW-Authenticate: Basic" response is returned and the invocation is not routed to the policy engine. As

a result, no events are recorded for that invocation, and its key performance indicator (KPI) data are not included in the performance metrics.

If none of the authentication actions ([HTTP Basic Authentication](#), [NTLM Authentication](#) or [OAuth2 Authentication](#)) is configured for a proxy API, Mediator forwards the request to the native API, without attempting to authenticate the request.

Input Parameters

Authenticate Using *String*. The user credentials for authenticating client requests to the native API.

<u>Value</u>	<u>Description</u>
Existing Credentials	<i>Default.</i> Mediator authenticates requests based on the credentials specified in the HTTP header. It passes the “Authorization” header present in the original client request to the native API.
Custom Credentials	Mediator authenticates requests according to the values you specify in the User , Password and Domain fields.
<u>Field</u>	<u>Description</u>
Username	<i>String. Mandatory.</i> Account name of a consumer who is available in the Integration Server on which Mediator is running.
Password	<i>String. Mandatory.</i> A valid password of the consumer.
Domain	<i>String. Optional.</i> Domain used by the server to authenticate the consumer.

Invoke webMethods Integration Server

Specifically, you would need to configure the *Invoke webMethods Integration Server* action to:

- Pre-process the request messages into the format required by the native API, before Mediator sends the requests to the native APIs.
- Pre-process the native API's response messages into the format required by the clients, before Mediator returns the responses to the clients.

In some cases an API might need to process messages.

For example, you might need to accommodate differences between the message content that a client is capable of submitting and the message content that a native API expects. For example, if the client submits an order record using a slightly different structure than the structure expected by the native API, you can use this action to process the record submitted by the client to the structure required by the native API.

In the Request Handling sequence, this action invokes the webMethods IS service to pre-process the request received from the client and before it is submitted to the native API.

In the Response Processing sequence, this action invokes the webMethods IS service to process the response message received from the native API and before it is returned to the client.

Note: A webMethods IS service must be running on the same Integration Server as Mediator. It can call out a C++ or Java or .NET function. It can also call other Integration Server services to manipulate the message.

Input Parameters

webMethods IS Service	<p><i>String. Mandatory.</i> Enter a name for the webMethods IS Service. This service will be used to manipulate the request/response (the axis2 MessageContext instance).</p> <p>Mediator will pass to the invoked IS service the request message context (the axis2 MessageContext instance), which contains the request-specific information. Also, you can use the public IS services that accept MessageContext as input to manipulate the response contents.</p>
-----------------------	--

JMS Routing Rule

This action allows you to specify a JMS queue to which the Mediator is to submit the request, and the destination to which the native API is to return the response.

To use the JMS Routing Rule action, you publish multiple APIs for a single native API. For example, to make a particular native API available to clients over both HTTP and JMS, you would create two APIs for the native API: one that accepts requests over HTTP

and another that accepts requests over JMS. Both APIs would route requests to the same native API on the back end.

Note: To make it easier to manage APIs, consider adopting a naming convention like the one shown above. Doing so will make it easier to identify APIs and the native API with which they are associated. Keep in mind however, that unlike native APIs, the names of APIs cannot contain spaces or special characters (except `_` and `-`). Consequently, if you adopt a convention that involves using the name of the native API as part of the API name, then the names of the native APIs themselves must not contain characters that are invalid in API names.

To use this action the following prerequisites must be met:

- Create an alias to a JNDI Provider (in the Integration Server Administrator, go to **Settings > Web Services**).
- To establish an active connection between Integration Server and the JMS provider, you must configure Integration Server to use a JMS connection alias (in the Integration Server Administrator, go to **Settings > Messaging > JMS Settings**).
- Create a WS (Web Service) endpoint alias for provider Web Service Descriptor (WSD) that uses a JMS binder. In the Integration Server Administrator, navigate to **Settings > Web Services** and complete the Alias Name, Description, Descriptor Type, and Transport Type fields.
- Configure a WS (Web Service) endpoint trigger (in the Integration Server Administrator, go to **Settings > Messaging > JMS Trigger Management**).
- Create a WS (Web Service) endpoint alias for consumer Web Service Descriptor (WSD) that has a JMS binder. In the Integration Server Administrator, navigate to **Settings > Web Services** and complete the Alias Name, Description, Descriptor Type, and Transport Type fields.
- Additionally, in the proxy API's **Message Flow** area, make sure that you delete the predefined *Straight Through Routing* and *HTTP Basic Authentication* actions from the **Receive** stage. This is because, these actions are mutually exclusive.

For detailed information and procedures, see the *webMethods Integration Server Administrator's Guide*.

Input Parameters

Connection URL	<i>String. Mandatory.</i> Specify a connection alias for connecting to the JMS provider (for example, an Integration Server alias or a JNDI URL). For example, a JNDI URL of the form: <code>jms:queue:dynamicQueues/MyRequestQueue? wm-wsendpointalias=MediatorConsumer &targetService=vs-jms-in-echo</code>
-------------------	---

Note that the `wm-wsendpointalias` parameter is required for Integration Server/Mediator to look up the JMS consumer

	alias to send the request to the specified queue (for example, <code>MyRequestQueue</code>), which is a dynamic queue in ActiveMQ. Also, the <code>targetService</code> parameter is required if sending to another API that uses JMS as the entry protocol.
Reply to Destination	<i>Optional.</i> Specify a queue name where a reply to a message should be sent.
Priority	Enter an integer that represents the priority of this JMS message with respect to other messages that are in the same queue. The priority value determines the order in which the messages are routed. The lower the Priority value, the higher the priority (that is, 0 is the highest priority, and messages with this priority value are executed first). <ul style="list-style-type: none"> ■ Priority values 0 through 10. ■ The default priority for a JMS message is 0.
Time to Live	<i>Optional.</i> A numeric value (in milliseconds) that specifies the expiration time of the JMS message. If the time-to-live is specified as zero, expiration is set to zero which indicates the message does not expire. The default value is 0.
Delivery Mode	<i>Optional.</i> The type of message delivery to the endpoint.

Value	Description
Persistent	The message is stored by the JMS server before delivering it to the client.
Non-Persistent	<i>Default.</i> The message is not stored before delivery.

What Happens if a Queue Includes Multiple JMS Messages?

To determine the order in which to execute the JMS messages in a queue, Mediator examines each message's Priority setting.

The Priority setting contains a non-negative integer that indicates the JMS message's priority. A priority value of 0 represents the highest possible priority.

Note: A JMS message's **Priority** property is used *only* when there are multiple JMS messages to route in the queue. If the queue has only one message to route, the **Priority** property is ignored entirely.

When a queue includes multiple JMS messages, Mediator routes the messages serially, in priority order from lowest to highest (that is, it routes with message the *lowest* priority value first). Each messages in the queue is routed to completion before the next one begins.

If two or more messages have the same priority value, their order is indeterminate. Mediator will route these messages in serial fashion after all lower priority messages and before any higher priority messages. However, you cannot predict their order

Example

If Mediator were given the following JMS messages to route for an API:

JMS Message	Priority
JMS Message A	11
JMS Message B	25
JMS Message C	11
JMS Message D	0

It would route the messages in the following order:

JMS Message	Priority
JMS Message D	0
JMS Message A then JMS Message C (or vice versa) The order of these two messages cannot be controlled or predicted because they have the same priority.	11
JMS Message B	25

Load Balancing and Failover Routing

If you have a native API that is hosted at two or more endpoints, you can use the Load Balancing and Failover Routing to distribute requests among the endpoints.

Requests are distributed across multiple endpoints. The requests are intelligently routed based on the "round-robin" execution strategy. The load for a service is balanced by directing requests to two or more services in a pool, until the optimum level is achieved. The application routes requests to services in the pool sequentially, starting from the first to the last service without considering the individual performance of the services.

After the requests have been forwarded to all the services in the pool, the first service is chosen for the next loop of forwarding.

Load-balanced endpoints also have automatic Failover capability. If a load-balanced endpoint is unavailable (for example, if a connection is refused), then that endpoint is marked as "down" for the number of seconds you specify in the `Timeout` field (during which the endpoint will not be used for sending the request), and the next configured endpoint is tried. If all the configured load-balanced endpoints are down, then a failure is sent back to the client. After the timeout expires, each endpoint marked will be available again to send the request.

Input Parameters

Route To *URI. Mandatory.* Enter the URLs of two or more endpoints in a pool to which the requests will be routed. The application routes the requests to endpoints in the pool sequentially, starting from the first to the last endpoint without considering the individual performance of the endpoints. After the requests have been forwarded to all the endpoints in the pool, the first endpoint is chosen for the next loop of forwarding.

Enter the URL of the endpoint to route the request to. For example:

```
http://mycontainer/creditCheckService
```

To specify additional endpoints, use the plus button next to the field to add rows.

Click the **Configure Endpoint Properties** icon (next to the field) if you want to configure a set of properties for the endpoints individually.

Alternatively, Mediator offers "Local Optimization" capability if the native API is hosted on the same Integration Server as Mediator. With local optimization, API invocation happens in-memory and not through a network hop.

```
local://<Service-full-path>
```

OR

```
local://<server>:<port>/ws/<Service-full-path>
```

For example:

```
local://MyAPIFolder:MyLocalAPI
```

which points to the endpoint API `MyLocalAPI` which is present under the folder `MyAPIFolder` in Integration Server.

Note: Local Optimization is not applicable to REST based APIs.

Configure Endpoint Properties 
(icon)

Optional. This icon displays the **Endpoint Properties** dialog box that enables you to configure a set of properties for the Mediator to route incoming requests to the native API as follows:

SOAP Optimization Method *Only for SOAP-based APIs.* Mediator can use the following optimization methods to parse SOAP requests to the native API:

Value	Description
MTOM	Mediator will use the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API.
SwA	Mediator will use the SOAP with Attachment (SwA) technique to parse SOAP requests to the API.
None	<i>Default.</i> Mediator will not use any optimization method to parse the SOAP requests to the API.

Note: Keep the following points in mind:

- Bridging between SwA and MTOM is not supported. If a client sends a SwA request, Mediator can only forward SwA to the native API. The same is true for MTOM, and applies to responses received from the native API. That is, a SwA or MTOM response received by Mediator from a native API will be forwarded to the client using the same format it received.
- When sending SOAP requests that do not contain a MTOM or SWA attachment to a native API that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related'. This is necessary so Mediator knows how to parse the response properly.

HTTP Connection Timeout

Number.Optional. The time interval (in seconds) after which a connection attempt will timeout. If a value 0 is specified (or if the value is not specified), Mediator will use the value specified

in the `Connection Timeout` field (in the **Integration Server Administrator**, go to **Settings > Extended**). Default: 30 seconds.

Read
Timeout

Number.Optional. The time interval (in seconds) after which a socket read attempt will timeout. If a value 0 is specified (or if the value is not specified), Mediator will use the value specified in the `Read Timeout` field (in the **Integration Server Administrator**, go to **> Settings > Extended**). Default: 30 seconds.

SSL
Configuration

Optional. To enable SSL client authentication that Mediator will use to authenticate incoming requests for the native API, you must specify values for both the `Client Certificate Alias` field and the `IS Keystore Alias` field. If you specify a value for only one of these fields, a deployment error will occur.

Note: SSL client authentication is optional; you may leave both fields blank.

Prerequisite: You must set up the key alias and keystore properties in the **Integration Server**. For the procedure, see *webMethods Integration Server Administrator's Guide*.

You will use these properties to specify the following fields:

Value	Description
Client Certificate Alias	<i>Mandatory.</i> The client's private key to be used for performing SSL client authentication.
IS Keystore Alias	<i>Mandatory.</i> The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of <code>Client Certificate Alias</code>) will be used for performing SSL client authentication.

WS Security Header *Only for SOAP-based APIs.* Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native API.

<u>Value</u>	<u>Description</u>
Remove processed security headers	<i>Default.</i> Removes the security header if it is processed by Mediator (that is, if Mediator processes the header according to the API's security run-time policy). Note that Mediator will not remove the security header if both of the following conditions are true: 1) Mediator did not process the security header, and 2) the mustUnderstand attribute of the security header is 0/false).
Pass all security headers	Passes the security header, even if it is processed by Mediator (that is, even if Mediator processes the header according to the API's security action).

Log Invocation

This action logs request/response payloads. You can specify the log destination and the logging frequency. This action also logs other information about the requests/responses, such as the API name, operation name, the Integration Server user, a timestamp, and the response time.

Input Parameters

Payloads *String.* Specify whether to log all request/response payloads.

<u>Value</u>	<u>Description</u>
Request	Logs all request payloads.
Response	Logs all response payloads.

Log Generation Frequency *String.* Specify how frequently to log the payload.

Value	Description
Always	Logs all requests and/or responses.
On Success	Logs only the successful responses and/or requests.
On Failure	Logs only the failed requests and/or responses.
Send Data To	<p data-bbox="431 638 951 667"><i>String</i>. Specify where to log the payload.</p> <div data-bbox="448 699 1349 800" style="background-color: #f0f0f0; padding: 5px;"> <p data-bbox="448 699 1349 800">Important Ensure that Mediator is configured to log the payloads to the destination(s) you specify here. For details, see <i>Administering webMethods Mediator</i>.</p> </div>
Value	Description
CentraSite	<p data-bbox="626 926 1240 989">Logs the payloads in the API's Events profile in CentraSite.</p> <p data-bbox="626 1014 1273 1186">Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to Solutions > Mediator > Administration > CentraSite Communication). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
Local Log	<p data-bbox="626 1234 1312 1297">Logs the payloads in the server log of the Integration Server on which Mediator is running.</p> <p data-bbox="626 1318 1187 1348">Also choose a value in the Log Level field:</p> <ul data-bbox="626 1369 1333 1545" style="list-style-type: none"> <li data-bbox="626 1369 1243 1432">■ Info: logs the error-level, warning-level, and informational-level alerts. <li data-bbox="626 1457 1333 1486">■ Warn: logs the error-level and warning-level alerts. <li data-bbox="626 1512 1101 1541">■ Error: logs only error-level alerts. <div data-bbox="643 1577 1365 1719" style="background-color: #f0f0f0; padding: 5px;"> <p data-bbox="643 1577 1365 1719">Important The Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to Settings > Logging > Server Logger).</p> </div>
SNMP	<p data-bbox="626 1759 1292 1822">Logs the payloads in CentraSite's SNMP server or a third-party SNMP server.</p> <p data-bbox="626 1843 1305 1908">Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator,</p>

	go to Solutions > Mediator > Administration > SNMP). For the procedure, see <i>Administering webMethods Mediator</i> .
Email	<p>Sends the payloads to an SMTP email server, which sends them to the email address(es) you specify here. Mediator sends the payloads as email attachments that are compressed using gzip data compression. To specify multiple addresses, use the plus button to add rows.</p> <p>Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
Audit Log	<p>Logs the payload to the Integration Server audit logger. For more information about logging, see the <i>webMethods Audit Logging Guide</i>.</p> <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p>Note: If you expect a high volume of events in your system, it is recommended that you select the Audit Log destination for this action.</p> </div>
EDA	<p>Mediator can use EDA to log the payloads to a database.</p> <p>Prerequisite: You must configure the EDA destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > EDA). For the procedure, see <i>Administering webMethods Mediator</i>.</p>

Monitor Service Level Agreement

This action monitors a set of run-time performance conditions for an API, and sends alerts to a specified destination when the performance conditions are violated. This action enables you to monitor run-time performance for *one or more specified clients*.

You can configure this action to define a *Service Level Agreement (SLA)*, which is a set of conditions that defines the level of performance that a client should expect from an API. You can use this action to identify whether an API threshold rules are met or exceeded. For example, you might define an agreement with a particular client that sends an alert to the client (consumer application) if responses are not sent within a certain maximum response time. You can configure SLAs for each API/consumer application combination.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), Mediator sends an alert as soon as the performance condition is violated, without having to wait until the end of the metrics tracking interval. You can choose whether to send an alert only once during the interval, or every time the violation occurs during the interval. (Mediator will send another alert the next time a condition is violated during

a subsequent interval.) For information about the metrics tracking interval, see "[The Metrics Tracking Interval](#)" on page 332.

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), Mediator aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

This action does not include metrics for failed invocations.

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see *Administering webMethods Mediator*.

Input Parameters

Action Configuration Parameters Specify one or more conditions to monitor. To do this, specify a metric, operator, and value for each metric. To specify multiple conditions, use the plus button to add multiple rows. If multiple parameters are used, they are connected by the AND operator.

NameOperator *String. Array.* The metrics to monitor.

Value	Description
Availability	Indicates whether the API was available to the specified clients in the current interval.
Average Response Time	The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment Mediator receives the request until the moment it returns the response to the caller.
Fault Count	Indicates the number of faults returned in the current interval.
Maximum Response Time	The maximum amount of time to respond to a request in the current interval.
Minimum Response Time	The minimum amount of time to respond to a request in the current interval.

	Successful Request Count	The number of successful requests in the current interval.
	Total Request Count	The total number of requests (successful and unsuccessful) in the current interval.
		<i>String. Array.</i> Specifies an operator.
Value		<i>Integer. Array.</i> Specifies an alert value.
Alert for Consumer Applications		<i>Object. Array.</i> Specify the Application asset(s) to which this Service Level Agreement will apply. To specify multiple consumer applications, use the plus button to add multiple rows.
Alert Configuration Parameters		Specifies the parameters for the alerts that will report on the Service Level Agreement conditions:
Alert Interval		<i>Number.</i> The time period (in minutes) in which to monitor performance before sending an alert if a condition is violated. For information about the metrics tracking interval, see " The Metrics Tracking Interval " on page 332.
Alert Frequency		<i>String.</i> Specifies how frequently to issue alerts for the counter-based metrics (Total Request Count, Success Count, Fault Count).
	Value	Description
	Every Time	Issue an alert every time one of the specified conditions is violated.
	Only Once	Issue an alert only the first time one of the specified conditions is violated.
Alert Destination		<i>String.</i> Specifies where to log the alert.
	Important Ensure that Mediator is configured to send event notifications to the destination(s) you specify here. For details about alerts and transaction logging, see <i>Administering webMethods Mediator</i> .	

Value	Description
CentraSite	<p>Sends the alerts to the API's Events profile in CentraSite.</p> <p>Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to Solutions > Mediator > Administration > CentraSite Communication). For information about how to configure communication with CentraSite, see <i>Administering webMethods Mediator</i>.</p>
Local Log	<p>Sends the alerts to the server log of the Integration Server on which Mediator is running.</p> <p>Also choose a value in the <code>Log Level</code> field:</p> <ul style="list-style-type: none"> ■ Info: Logs error-level, warning-level, and informational-level alerts. ■ Warn: Logs error-level and warning-level alerts. ■ Error: Logs only error-level alerts. <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Important The Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to Settings > Logging > Server Logger).</p> </div>
SNMP	<p>Sends the alerts to CentraSite's SNMP server or a third-party SNMP server.</p> <p>Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
Email	<p>Sends the alerts to an SMTP email server, which sends them to the email address(es) you specify here. To specify multiple addresses, use the plus button to add rows.</p> <p>Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to Solutions > Mediator ></p>

Administration > Email). For the procedure, see *Administering webMethods Mediator*.

EDA

Mediator can use EDA to log the payloads to a database.

Prerequisite: You must configure the EDA destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > EDA**). For the procedure, see *Administering webMethods Mediator*.

Alert Message *String. Optional.* Specify a text message to include in the alert.

Monitor Service Performance

This action is similar to the [Monitor Service Level Agreement](#) action. Both actions can monitor the same set of run-time performance conditions for an API, and then send alerts when the performance conditions are violated. However, this action monitors run-time performance for *a specific client*.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), Mediator sends an alert as soon as the performance condition is violated, without having to wait until the end of the metrics tracking interval. You can choose whether to send an alert only once during the interval, or every time the violation occurs during the interval. (Mediator will send another alert the next time a condition is violated during a subsequent interval.) For information about the metrics tracking interval, see "[The Metrics Tracking Interval](#)" on page 332.

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), Mediator aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

This action does not include metrics for failed invocations.

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For detailed information about configuring communication with CentraSite, see *Administering webMethods Mediator*.

Input Parameters

Action Configuration Parameters	Specify one or more conditions to monitor. To do this, specify a metric, operator, and value for each metric. To specify multiple conditions, use the plus button to add multiple rows. If multiple parameters are used, they are connected by the AND operator.
---------------------------------	--

NameOperator *String. Array.* The metrics to monitor.

<u>Value</u>	<u>Description</u>
Availability	Indicates whether the service was available to the specified clients in the current interval.
Average Response Time	The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment Mediator receives the request until the moment it returns the response to the caller.
Fault Count	Indicates the number of faults returned in the current interval.
Maximum Response Time	The maximum amount of time to respond to a request in the current interval.
Minimum Response Time	The minimum amount of time to respond to a request in the current interval.
Successful Request Count	The number of successful requests in the current interval.
Total Request Count	The total number of requests (successful and unsuccessful) in the current interval.

String. Array. Specify an operator.

Value *Integer. Array.* Specify an alert value.

Alert Configuration Parameters Specify the parameters for the alerts that will report on the Service Level Agreement conditions:

Alert Interval *Number.* The time period (in minutes) in which to monitor performance before sending an alert if a condition is violated. For information about the metrics tracking interval, see "[The Metrics Tracking Interval](#)" on page 332.

Alert
Frequency

String. Specify how frequently to issue alerts for the counter-based metrics (Total Request Count, Success Count, Fault Count).

<u>Value</u>	<u>Description</u>
Every Time	Issue an alert every time one of the specified conditions is violated.
Only Once	Issue an alert only the first time one of the specified conditions is violated.

Alert
Destination

String. Specify where to log the alert.

Important Ensure that Mediator is configured to send event notifications to the destination(s) you specify here. For details about alerts and transaction logging, see *Administering webMethods Mediator*.

<u>Value</u>	<u>Description</u>
CentraSite	<p>Sends the alerts to the API's Events profile in CentraSite.</p> <p>Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to Solutions > Mediator > Administration > CentraSite Communication). For information about how to configure communication with CentraSite, see <i>Administering webMethods Mediator</i>.</p>
Local Log	<p>Sends the alerts to the server log of the Integration Server on which Mediator is running.</p> <p>Also choose a value in the Log Level field:</p> <ul style="list-style-type: none"> ■ Info: Logs error-level, warning-level, and informational-level alerts. ■ Warn: Logs error-level and warning-level alerts. ■ Error: Logs only error-level alerts. <p>Important Integration Server Administrator's logging level for Mediator should match</p>

the logging level specified for this action (go to **Solutions > Logging > Server Logger**).

SNMP	<p>Sends the alerts to CentraSite's SNMP server or a third-party SNMP server.</p> <p>Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
Email	<p>Sends the alerts to an SMTP email server, which sends them to the email address(es) you specify here. To specify multiple addresses, use the plus button to add rows.</p> <p>Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
EDA	<p>Mediator can use EDA to log the payloads to a database.</p> <p>Prerequisite: You must configure the EDA destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > EDA). For the procedure, see <i>Administering webMethods Mediator</i>.</p>

Alert Message *String. Optional.* Specify a text message to include in the alert.

NTLM Authentication

This action uses the NTLM authentication to validate incoming requests from clients. Mediator authorizes the NTLM credentials (username and password) against a list of all global consumers available in the Mediator.

If the username/password value in the Authorization header cannot be authenticated as a valid Integration Server user (or if the Authorization header is not present in the request), a 500 SOAP fault is returned, and the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. If the client does not successfully respond to the challenge, a 401 Unauthorized "WWW-Authenticate: NTLM" (for NTLM authentication) or "WWW-Authenticate: Negotiate" (for Kerberos authentication) is returned in the response header and the invocation is not routed to the policy engine. As a result, no events are recorded for

that invocation, and its key performance indicator (KPI) data are not included in the performance metrics.

Note: Note that if Mediator is used to access a native API protected by NTLM (which is typically hosted in IIS), then the native API in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list.

If none of the authentication actions ([HTTP Basic Authentication](#), [NTLM Authentication](#) or [OAuth2 Authentication](#)) is configured for a proxy API, Mediator forwards the request to the native API, without attempting to authenticate the request.

Input Parameters

Authenticate Using *String*. Specifies the user credentials for authenticating client requests to the native API.

Note: If Mediator is used to access a native API protected by NTLM (which is typically hosted in IIS), then the native API in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM.

Value	Description
Existing Credentials	<i>Default.</i> Mediator uses the user credentials passed in the request header for an NTLM handshake with the server.
Custom Credentials	Mediator uses the values you specify in the User , Password and Domain fields for an NTLM handshake with the server.

Field	Description
Username	<i>String. Mandatory.</i> Account name of a consumer who is available in the Integration Server on which Mediator is running.
Password	<i>String. Mandatory.</i> A valid password of the consumer.

	Domain	<i>String. Optional.</i> Domain used by the server to authenticate the consumer.
Transparent	Mediator supports Kerberos handshake in Transparent mode. The following additional settings are required for Kerberos:	<ul style="list-style-type: none"> ■ Configure the client with <code>clientCredentialType</code> set to <code>Windows</code>. ■ Set the value of <code>watt.pg.disableNtlmAuthHandler</code> property to <code>true</code> in the extended settings for the Integration Server ■ Set the property <code>handleClientErrorCode</code> to <code>true</code> in <code>pg-core.xml</code> as follows:

```
<bean
  id="httpResponseCodeCallback"
  class="com.softwareag.pg.axis2.transpo
rts.ISHTTPResponseCodeCallback">
<property name="handleClientErrorCode"
  value="true"/>
</bean>
```

For more information about configuring the extended settings, see *webMethods Integration Server Administrator's Guide*.

OAuth2 Authentication

This action uses the OAuth 2.0 authentication to validate incoming requests from clients. Mediator authorizes the OAuth 2.0 credentials (access token) against a list of all global consumers available in the Mediator.

If the access token value in the Authorization header cannot be authenticated as a valid Integration Server user (or if the Authorization header is not present in the request), a 500 SOAP fault is returned, and the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. If the client does not successfully respond to the challenge, a "WWW-Authenticate: OAuth" response is returned and the invocation is not routed to the policy engine. As a result, no events are recorded for that invocation, and its key performance indicator (KPI) data are not included in the performance metrics.

If none of the authentication actions ([HTTP Basic Authentication](#), [NTLM Authentication](#) or [OAuth2 Authentication](#)) is configured for a proxy API, Mediator forwards the request to the native API, without attempting to authenticate the request.

Input Parameters

Authenticate Using *String*. Specifies the OAuth2 access token for authenticating client requests to the native API.

Value	Description
Existing Token	<i>Default.</i> Mediator uses the OAuth2 access token specified in the HTTP "Authorization" header to validate client requests for a native API.
Custom Token	Mediator uses the access token you specify in the OAuth2 Token , field to validate client requests for a native API.

Field	Description
OAuth2 Token	<i>String. Mandatory.</i> Specifies an OAuth2 access token to be deployed by Mediator. The consumer need not pass the OAuth2 token during service invocation.

Response Transformation

The *Response Transformation* action specifies:

- The XSLT transformation file to transform response messages from native APIs into a format required by the client.

In some cases a message needs to be transformed prior to sending to the client.

For example, you might need to accommodate differences between the message content that a native API is capable of submitting and the message content that a client expects. For example, if the native API submits an order record using a slightly different structure than the structure expected by the client, you can use this action to transform the record submitted by the native API to the structure required by the client.

When this action is configured for a proxy API, the native API's response messages are transformed into the format required by the client, before Mediator returns the responses to the clients.

Input Parameters

Transformation File *File. Mandatory.* Click **Choose**, select the XSL transformation file from your file system and click **OK**.

When you virtualize an API, the transformation file is validated. If there are no validation errors, the XSLT file is displayed as a download link in the same dialog. If the transformation file is invalid (for example, non-XSLT file), this will be indicated by a warning icon.

Note: If you make changes to the XSLT file in the future, you must republish the API.

Important: The XSL file uploaded by the user should not contain the XML declaration in it (for example, `xml version="1.0" encoding="UTF-8"`). This is because when the API is published to Mediator, Mediator embeds the XSL file in the virtual service definition (VSD), and since the VSD itself is in XML format, there cannot be an XML declaration line in the middle of it. This can lead to unexpected deployment issues which can be avoided by making sure the XSL file does not contain the declaration line.

Request Transformation

The *Request Transformation* action specifies:

- The XSLT Transformation File to transform request messages from clients into a format required by the native API.

In some cases a native API might need to transform messages.

For example, you might need to accommodate differences between the message content that a client is capable of submitting and the message content that a native API expects. For example, if the client submits an order record using a slightly different structure than the structure expected by the native API, you can use this action to transform the record submitted by the client to the structure required by the native API.

When this action is configured for a proxy API, the incoming requests from the clients are transformed into a format required by the native API, before Mediator sends the requests to the native APIs.

Input Parameters

Transformation File *File. Mandatory.* Click **Choose**, select the XSL transformation file from your file system and click **OK**.

When you virtualize an API, the transformation file is validated. If there are no validation errors, the XSLT file is displayed as a download link in the same dialog. If the transformation file is invalid (for example, non-XSLT file), this will be indicated by a warning icon.

Note: If you make changes to the XSLT file in the future, you must republish the API.

Important: The XSL file uploaded by the user should not contain the XML declaration in it (for example, `xml version="1.0" encoding="UTF-8"`). This is because when the API is published to Mediator, Mediator embeds the XSL file in the virtual service definition (VSD), and since the VSD itself is in XML format, there cannot be an XML declaration line in the middle of it. This can lead to unexpected deployment issues which can be avoided by making sure the XSL file does not contain the declaration line.

Require Encryption

This action requires that a request's XML element (which is represented by an XPath expression) be encrypted.

To use this action, the following prerequisites must be met:

1. Configure Integration Server: Set up keystores and truststores in Integration Server, as described in the *webMethods Integration Server Administrator's Guide*.
2. Configure Mediator: In the Integration Server Administrator, navigate to **Solutions > Mediator > Administration > General** and complete the IS Keystore Name, IS Truststore Name and Alias (signing) fields, as described in *Administering webMethods Mediator*.

When this action is configured for a proxy API, Mediator provides decryption of incoming requests and encryption of outgoing responses. Mediator can encrypt and decrypt only individual elements in the SOAP message body that are defined by the XPath expressions configured for the action. Mediator requires that requests contain the encrypted elements that match those in the XPath expression. You must encrypt the entire element, not just the data between the element tags. Mediator rejects requests if the element name is not encrypted.

Important: Do not encrypt the entire SOAP body because a SOAP request without an element will appear to Mediator to be malformed.

Mediator attempts to encrypt the response elements that match the XPath expressions with those defined for the action. If the response does not have any elements that match the XPath expression, Mediator will not encrypt the response before sending. If the XPath expression resolves a portion of the response message, but Mediator cannot locate

a certificate to encrypt the response, then Mediator sends a SOAP fault exception to the client and a Policy Violation event notification to CentraSite.

How Mediator Encrypts Responses

The Require Encryption action encrypts the response back to the client by dynamically setting a public key alias at run time. Mediator determines the public key alias as follows:

- If Mediator can access the X.509 certificate of the client (based on the incoming request signature), it will use “useReqSigCert” as the public key alias.

OR

- If an “Evaluate” action is present in the message flow (and it successfully identifies a client), then Mediator will look for a public key alias with that client name in the “IS Keystore Name” property. The “IS Keystore Name” property is specified in the Integration Server Administrator, under **Solutions > Mediator > Administration > General**. This property should be set to an Integration Server keystore that Mediator will use.

For an “Evaluate” action that allows for anonymous usage, Mediator does *not* require a client name in order to send encrypted responses. In this case, Mediator can use one of the following to encrypt the response in the following order, depending on what is present in the security element:

- A signing certificate.
- Client name.
- WSS username, SAML token or X.509 certificate.
- HTTP authorized user.

OR

- If Mediator can determine the current IS user from the request (that is, if an Integration Server WS-Stack determined that Subject is present), then the first principal in that subject is used.

OR

- If the above steps all fail, then Mediator will use either the WS-Security username token or the HTTP Basic-Auth username value. There should be a public key entry with the same name as the identified username.

Input Parameters

Namespace

String. Mandatory. Namespace of the element required to be encrypted.

Note: Enter the namespace prefix in the following format:
`xmlns:<prefix-name>`. For example: `xmlns:soapenv`.

The generated XPath element in the policy should look similar to this:

```
<sp:SignedElements xmlns:sp=
"http://docs.oasis-open.org/
ws-sx/ws-securitypolicy/200702">
<sp:XPath xmlns:soapenv=
"http://schemas.xmlsoap.org/soap/envelope
/">//soapenv:Body</sp:XPath>
</sp:SignedElements>
```

Element to be
Encrypted

String. Mandatory. An XPath expression that represents the XML element that is required to be encrypted. See the sample below.

Let's take a look at an example. For the following SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
</soap:Header>
  <soap:Body>
    <catalog xmlns="http://www.store.com">
      <name>My Book</name>
      <author>ABC</author>
      <price>100</price>
    </catalog>
  </soap:Body>
</soap:Envelope>
```

The XPath expression appears as follows:

```
/soap:Envelope/soap:Body
```

Require HTTP / HTTPS

If you have a native API that requires clients to communicate with the server using the HTTP and/or HTTPS protocols, you can use the Require HTTP / HTTPS protocol action.

This action allows you to bridge the transport protocols between the client and the Mediator. For example, suppose you have a native API that is exposed over HTTPS and an API that receives requests over HTTP. In this situation, you can configure the API's Require HTTP / HTTPS action to accept HTTP requests and configure its Routing action to route the request to the native API using HTTPS.

Input Parameters

Protocol *String.* Specifies the protocol over which the Mediator accepts requests from the client.

Note: CentraSite supports HTTP version 1.1 only.

Important: Before you deploy an API over HTTPS, ensure that the Integration Server on which the Mediator is running has been configured for SSL. In addition, make sure you specify

an HTTPS port in the Mediator's `Ports Configuration` page. (In the Integration Server Administrator, go to **Solutions > Mediator > Administration > General** and specify the port in the **HTTPS Ports Configuration** field.) For details on the Port Configuration page, see *Administering webMethods Mediator*.)

<u>Value</u>	<u>Description</u>
HTTP	<i>Default.</i> Mediator will only accept requests that are sent using the HTTP protocol.
HTTPS	Mediator will only accept requests that are sent using the HTTPS protocol.

You can select *both* HTTP and HTTPS if needed.

SOAP
Version *String.* For SOAP-based APIs. Specifies the SOAP version of the requests which the Mediator accepts from the client.

<u>Value</u>	<u>Description</u>
SOAP 1.1	<i>Default.</i> Mediator will only accept requests that are in the SOAP 1.1 format.
SOAP 1.2	Mediator will only accept requests that are in the SOAP 1.2 format.

Require JMS

If you have a native API that requires clients to communicate with the server using the JMS protocol, you can use the Require JMS protocol action.

This action allows you to bridge protocols between the client and the native API. For example, suppose you have a native API that is exposed over JMS and a client that submits SOAP requests over HTTP. In this situation, you can configure the API's Require JMS Protocol action to accept SOAP requests over Java Message Service (JMS) and configure its JMS Routing Rule action to route the request to the Web service using JMS.

When this action is configured for a proxy API, you can intentionally expose an API over a JMS protocol. For example, if you have a native API that is exposed over HTTP, you might expose the API over JMS simply to gain the asynchronous-messaging and guaranteed-delivery benefits that one gains by using JMS as the message transport.

To use this action the following prerequisites must be met:

- Create an alias to a JNDI Provider (in the Integration Server Administrator, go to **Settings > Web Services**).
- To establish an active connection between Integration Server and the JMS provider, you must configure Integration Server to use a JMS connection alias (in the Integration Server Administrator, go to **Settings > Messaging > JMS Settings**).
- Create a WS (Web Service) endpoint alias for provider Web Service Descriptor (WSD) that uses a JMS binder. In the Integration Server Administrator, navigate to **Settings > Web Services** and complete the Alias Name, Description, Descriptor Type, and Transport Type fields.
- Configure a WS (Web Service) endpoint trigger (in the Integration Server Administrator, go to **Settings > Messaging > JMS Trigger Management**).
- Create a WS (Web Service) endpoint alias for consumer Web Service Descriptor (WSD) that has a JMS binder. In the Integration Server Administrator, navigate to **Settings > Web Services** and complete the Alias Name, Description, Descriptor Type, and Transport Type fields.
- Additionally, in the API's Message Flow, make sure that you delete the predefined *Require HTTP / HTTPS Protocol* action from the Receive stage. This is because, these actions are mutually exclusive.

For detailed procedures, see *webMethods Integration Server Administrator's Guide*.

Input Parameters

JMS Provider Alias *String. Mandatory.* Specify the name of Integration Servers JMS provider alias. The alias should include the JNDI destination name and the JMS connection factory.

SOAP Version *String.* Specify the SOAP version of the requests which the Mediator accepts from the client.

Value	Description
SOAP 1.1	<i>Default.</i> Mediator will only accept requests that are in the SOAP 1.1 format.
SOAP 1.2	Mediator will only accept requests that are in the SOAP 1.2 format.

Require Signing

This action requires that a request's XML element (which is represented by an XPath expression) be signed.

Prerequisites

1. Configure Integration Server: Set up keystores and truststores in Integration Server, as described in the *webMethods Integration Server Administrator's Guide*.
2. Configure Mediator: In the Integration Server Administrator, navigate to **Solutions > Mediator > Administration > General** and complete the IS Keystore Name, IS Truststore Name and Alias (signing) fields, as described in *Administering webMethods Mediator*. Mediator uses the signing alias specified in the Alias (signing) field to sign the response.

When this action is configured for a proxy API, Mediator validates that the requests are properly signed, and provides signing for responses. Mediator provides support both for signing an entire SOAP message body or individual elements of the SOAP message body. Mediator uses a digital signature element in the security header to verify that all elements matching the XPath expression were signed. If the request contains elements that were not signed or no signature is present, then Mediator rejects the request.

Note: You must map the public certificate of the key used to sign the request to an Integration Server user. If the certificate is not mapped, Mediator returns a SOAP fault to the caller.

Input Parameters

Namespace *String. Mandatory.* Namespace of the element required to be signed.

Note: Enter the namespace prefix in the following format:
xmlns:<prefix-name>. For example: xmlns:soapenv.

The generated XPath element in the policy should look similar to this:

```
<sp:SignedElements xmlns:sp=
"http://docs.oasis-open.org/ws-sx/
ws-securitypolicy/200702">
<sp:XPath xmlns:soapenv=
"http://schemas.xmlsoap.org/soap/envelope
/">//soapenv:Body</sp:XPath>
</sp:SignedElements>
```

Element to be Signed *String. Mandatory.* An XPath expression that represents the XML element that is required to be signed. See the sample below.

Let's take a look at an example. For the following SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
</soap:Header>
  <soap:Body>
    <catalog xmlns="http://www.store.com">
      <name>My Book</name>
```

```

    <author>ABC</author>
    <price>100</price>
  </catalog>
</soap:Body>
</soap:Envelope>

```

The XPath expression appears as follows:

```
/soap:Envelope/soap:Body
```

Require SSL

This action requires that requests be sent via SSL client certificates.

When this action is configured for a proxy API, Mediator ensures that requests are sent to the server using the HTTPS protocol (SSL). The action also specifies whether the client certificate is required. This allows Mediator to verify the client sending the request. If the action requires the client certificate, but it is not presented, Mediator rejects the message.

When a client certificate is required by the action, the Integration Server HTTPS port should be configured to request or require a client certificate.

Note: In Integration Server, create an HTTPS port, as described in the *webMethods Integration Server Administrator's Guide*.

Input Parameters

Client Certificate Required	<p>Specifies whether client certificates are required for the purposes of:</p> <ul style="list-style-type: none"> ■ Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests ■ Signing SOAP responses or encrypting SOAP responses
-----------------------------	---

Require Timestamps

When this action is set for the API, Mediator requires that timestamps be included in the request header. Mediator checks the timestamp value against the current time to ensure that the request is not an old message. This serves to protect your system against attempts at message tampering, such as replay attacks.

Mediator rejects the request if either of the following happens:

- Mediator receives a timestamp that exceeds the time defined by the timestamp element.
- A timestamp element is not included in the request.

Note: This action has no input parameters; you simply drag and drop the action into the **Message Flow** area.

Input Parameters

None.

Require WSS SAML Token

When this action is configured for a proxy API, Mediator uses a WSS Security Assertion Markup Language (SAML) assertion token to validate clients for an API.

Note: For information about configuring your system for SAML token processing, see *Administering webMethods Mediator*.

Input Parameters

SAML Subject
Confirmation

String. Specifies the SAML subject confirmation methods:

Value	Description
Holder of Key	<p>Default. Select this option if clients use the SAML V1.1 or V2.0 Holder-of-Key Web Browser SSO Profile, which allows for transport of holder-of-key assertions. In this scenario, the client presents a holder-of-key SAML assertion acquired from its preferred identity provider to access a web-based resource at an API provider.</p> <p>If you select <code>Holder of Key</code>, Mediator also implicitly selects the “timestamp” and “signing” assertions to the virtual service definition (VSD). Thus, you should not add the “Require Timestamps” and “Require Signing” actions to the API if the “Require WSS SAML Token” action is already applied.</p>
Bearer	<p>Select this option if clients use the SAML V1.1 Bearer token authentication, in which a Bearer token mechanism relies upon bearer semantics as a means by which the client conveys to Mediator the sender's identity.</p> <p>If you select <code>Bearer</code>, the “timestamp” and “signing” assertions will be added to the virtual service definition (VSD).</p>

Note: If clients use SAML 2.0 Sender-Vouches tokens, configure your system as

described in *Administering WebMethods Mediator*.

SAML Version *String*. Specifies the WSS SAML Token version to use: 1.1 or 2.0.

Set Custom Headers

When this action is configured for a proxy API, Mediator includes custom HTTP headers to the client requests before submitting to the native APIs.

Input Parameters

Header *String*. Mediator uses the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

<u>Value</u>	<u>Description</u>
Name	<i>String</i> . A name for the HTTP header field. The header field name (\$field) is not case sensitive.
Value	<i>String</i> . A value for the HTTP header field.

Sample

Let's imagine you have a **Name** field "Authorization". This will be encoded in Base64 scheme as follows: QXV0aG9yaXphdGlvbg==.

Set JMS Headers

Every JMS message includes message header properties that are always passed from provider to client. The purpose of the header properties is to convey extra information to the client outside the normal content of the message body.

When this action is configured for a proxy API, Mediator uses the JMS header properties to authenticate client requests before submitting to the native APIs.

To use this action the following prerequisites must be met:

- Create an alias to a JNDI Provider (in the Integration Server Administrator, go to **Settings > Web Services**).
- To establish an active connection between Integration Server and the JMS provider, you must configure Integration Server to use a JMS connection alias (in the Integration Server Administrator, go to **Settings > Messaging > JMS Settings**).
- Create a WS (Web Service) endpoint alias for provider Web Service Descriptor (WSD) that uses a JMS binder. In the Integration Server Administrator, navigate to

Settings > Web Services and complete the Alias Name, Description, Descriptor Type, and Transport Type fields.

- Configure a WS (Web Service) endpoint trigger (in the Integration Server Administrator, go to **Settings > Messaging > JMS Trigger Management**).
- Create a WS (Web Service) endpoint alias for consumer Web Service Descriptor (WSD) that has a JMS binder. In the Integration Server Administrator, navigate to **Settings > Web Services** and complete the Alias Name, Description, Descriptor Type, and Transport Type fields.

For detailed instructions, see *webMethods Integration Server Administrator's Guide*.

Input Parameters

Header *String*. The JMS message headers that Mediator will use to authenticate incoming requests for the native API. To add additional rows, use the plus button.

<u>Value</u>	<u>Description</u>
Name	<i>String</i> . A name for the JMS message header field. The header field name (\$field) is not case sensitive.
Value	<i>String</i> . A value for the JMS message header field.

Settable JMS Header Properties

<u>Property Name</u>	<u>Property Type</u>	<u>Getter Method</u>
Message ID	string	getJMSMessageID()
Priority	int	getJMSPriority()
Time To Live	long	getTimeToLive()
Delivery Mode	int	getJMSDeliveryMode()
Message Expiration	long	getJMSExpiration()
Correlation ID	string	getJMSCorralationID()
Redelivered	boolean	getJMSRedelivered()
Time Stamp	long	getJMSTimeStamp()

Property Name	Property Type	Getter Method
Type	string	getJMSType()

Set Media Type

This action specifies the content type for a REST request or response. You would need to configure the **Set Media Type** action to:

- Specify the content type for a REST request received from a client. If the content type header is missing in a client request sent to an API, Mediator adds the content type specified here before sending the request to the native API.
- Specify the content type for a REST response (both for a REST API and SOAP API exposed as REST) before sending the response to the client. If a native API response (both for a REST API and SOAP API exposed as REST) does not contain a content type header, Mediator adds the content type specified here before sending the response to the client.

Note: This action is applicable only if the action **Enable REST Support** is set for a SOAP API.

Input Parameters

Media Type *String*. The content type header that CentraSite would use to send the REST request or response.

Examples for content types: application/json, application/xml

Set Message Properties

The message property fields are similar to header fields described previously in the *Set JMS Headers* action, except these fields are set exclusively by the consumer application. When a client receives a message, the properties are in read-only mode. If a client tries to modify any of the properties, a `MessageNotWriteableException` will be thrown.

The properties are standard Java name/value pairs. The property names must conform to the message selector syntax specifications defined in the `Message` interface.

Property fields are most often used for message selection and filtering. By using a property field, a message consumer can interrogate the property field and perform message filtering and selection.

When this action is configured for a proxy API, Mediator uses the message properties to authenticate client requests before submitting to the native APIs.

To use this action the following prerequisites must be met:

- Create an alias to a JNDI Provider (in the Integration Server Administrator, go to **Settings > Web Services**).
- To establish an active connection between Integration Server and the JMS provider, you must configure Integration Server to use a JMS connection alias (in the Integration Server Administrator, go to **Settings > Messaging > JMS Settings**).
- Create a WS (Web Service) endpoint alias for provider Web Service Descriptor (WSD) that uses a JMS binder. In the Integration Server Administrator, navigate to **Settings > Web Services** and complete the Alias Name, Description, Descriptor Type, and Transport Type fields.
- Configure a WS (Web Service) endpoint trigger (in the Integration Server Administrator, go to **Settings > Messaging > JMS Trigger Management**).
- Create a WS (Web Service) endpoint alias for consumer Web Service Descriptor (WSD) that has a JMS binder. In the Integration Server Administrator, navigate to **Settings > Web Services** and complete the Alias Name, Description, Descriptor Type, and Transport Type fields.

For detailed information, see *webMethods Integration Server Administrator's Guide*.

Input Parameters

Property *String*. The custom message properties Mediator will use to authenticate incoming requests for the native API. To add additional rows, use the plus button.

Value	Description
Name	<i>String</i> . The name of the property.
Value	<i>String</i> . The value of the property.

Straight Through Routing

This action routes the incoming requests to the Mediator directly to the native API.

1. Configure Integration Server: Set up keystores and truststores in Integration Server, as described in the *webMethods Integration Server Administrator's Guide*.
2. Configure Mediator: In the Integration Server Administrator, navigate to **Solutions > Mediator > Administration > General** and complete the IS Keystore Name, IS Truststore Name and Alias (signing) fields, as described in *Administering webMethods Mediator*.

When this action is configured for an API, Mediator ensures that requests from the client are parsed directly to the native API you specify. This action also includes a set of configuration properties for the Mediator to process the incoming requests for the native API.

Input Parameters

Route To *URI. Mandatory.* Enter the URL of the native API endpoint to route the request to in case all routing rules evaluate to False. For example:

```
http://mycontainer/creditCheckService
```

Click the **Configure Endpoint Properties** icon (next to the **Route To** field) if you want to configure a set of properties for the specified endpoint.

Alternatively, Mediator offers "Local Optimization" capability if the native endpoint is hosted on the same Integration Server as Mediator. With local optimization, API invocation happens in-memory and not through a network hop.

Specify the native API in either of the following forms:

```
local://<Service-full-path>
```

OR

```
local://<server>:<port>/ws/<Service-full-path>
```

For example:

```
local://MyAPIFolder:MyLocalAPI
```

which points to the endpoint API `MyLocalAPI` which is present under the folder `MyAPIFolder` in Integration Server.

Note: Local Optimization is not applicable to REST based APIs.

Configure Endpoint Properties 
(icon)

Optional. This icon displays the **Endpoint Properties** dialog box that enables you to configure a set of properties for the Mediator to route incoming requests to the native API as follows:

SOAP Optimization Method	<i>For SOAP-based APIs.</i> Specifies the optimization methods that Mediator can use to parse SOAP requests to the native API.
--------------------------	--

Value	Description
MTOM	Mediator will use the Message Transmission Optimization Mechanism (MTOM) to parse SOAP requests to the API.
SwA	Mediator will use the SOAP with Attachment (SwA)

technique to parse SOAP requests to the API.

None

Default. Mediator will not use any optimization method to parse the SOAP requests to the API.

Note: Keep the following points in mind:

- Bridging between SwA and MTOM is not supported. If a client sends a SwA request, Mediator can only forward SwA to the native API. The same is true for MTOM, and applies to responses received from the native API. That is, a SwA or MTOM response received by Mediator from a native API will be forwarded to the client using the same format it received.
- When sending SOAP requests that do not contain a MTOM or SWA attachment to a native API that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related'. This is necessary so Mediator knows how to parse the response properly.

HTTP
Connection
Timeout

Number.Optional. Specifies the time interval (in seconds) after which a connection attempt will timeout. If a value 0 is specified (or if the value is not specified), Mediator will use the value specified in the `Connection Timeout` field (in the Integration Server Administrator, go to **Settings > Extended**). Default: 30 seconds.

Read Timeout

Number.Optional. Specifies the time interval (in seconds) after which a socket read attempt will timeout. If a value 0 is specified (or if the value is not specified), Mediator will use the value specified in the `Read Timeout` field (in the Integration Server Administrator, go to **Settings > Extended**). Default: 30 seconds.

SSL
Configuration

Optional. To enable SSL client authentication that Mediator will use to authenticate incoming requests for the native API, you must specify values for both the Client Certificate Alias field and the IS Keystore Alias field. If you specify a value for only one of these fields, a deployment error will occur.

Note: SSL client authentication is optional; you may leave both fields blank.

Prerequisite: You must set up the key alias and keystore properties in the Integration Server. For the procedure, see *webMethods Integration Server Administrator's Guide*.

You will use these properties to specify the following fields:

<u>Value</u>	<u>Description</u>
Client Certificate Alias	<i>Mandatory.</i> The client's private key to be used for performing SSL client authentication.
IS Keystore Alias	<i>Mandatory.</i> The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) will be used for performing SSL client authentication.

WS Security
Header

For SOAP-based APIs.

Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native API.

<u>Value</u>	<u>Description</u>
Remove processed security headers	<i>Default.</i> Removes the security header if it is processed by Mediator (that is, if Mediator processes the header according to the API's security run-time action). Note that

Mediator will *not* remove the security header if *both* of the following conditions are true: 1) Mediator did not process the security header, and 2) the `mustUnderstand` attribute of the security header is 0/false).

Pass all security headers

Passes the security header, even if it is processed by Mediator (that is, even if Mediator processes the header according to the API's security action).

Throttling Traffic Optimization

This action limits the number of API invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated.

Reasons for limiting the API invocation traffic include:

- To avoid overloading the back-end services and their infrastructure.
- To limit specific clients in terms of resource usage (that is, you can use the “Monitor Service Level Agreement” action to monitor performance conditions for a particular client, together with “Throttle API Usage” to limit the resource usage).
- To shield vulnerable servers, services, and even specific operations.
- For API consumption metering (billable pay-per-use APIs).

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see *Administering webMethods Mediator*.

Input Parameters

`Soft Limit` *Number.Optional.* The maximum number of invocations allowed per `Interval Value` before issuing an alert. Reaching the soft limit will not affect further processing of requests (until the `Hard Limit` is reached).

Note: The limit is reached when the total number of invocations coming from *all* the clients (specified in the `Limit Traffic for Applications` field) reaches the limit. `Soft Limit` is computed in an asynchronous manner; thus when multiple requests are made at the same time, it may

be possible that the Soft Limit alert will not be strictly accurate.

Alert Message for Soft Limit *String.Optional.* A text message to include in the soft limit alert.

Hard Limit **Required.** The maximum number of invocations allowed per `Interval Value` before stopping the processing of further requests and issuing an alert. Typically, this number should be higher than the soft limit.

Note: The limit is reached when the total number of invocations coming from *all* the clients (specified in the `Limit Traffic for Consumers` field) reaches the limit. Hard Limit is computed in an asynchronous manner; thus when multiple requests are made at the same time, it may be possible that the Hard Limit alert will not be strictly accurate.

Alert Message for Hard Limit *String.Optional.* A text message to include in the hard limit alert.

Alert for Consumer Applications *String.* The consumer application(s) that this action applies to. To specify multiple consumers, use the plus button to add rows, or select **Any Consumer** to apply this action to any consumer application.

Alert Interval *String.* The amount and unit (Minutes, Hours, Days or Weeks) of time for the soft limit and hard limit to be reached.

Alert Frequency *String.* Frequency to issue alerts.

Value	Description
Every Time	Default. Issues an alert every time the specified condition is violated.
Only Once	Issues an alert only the first time the specified condition is violated.

Alert Destination *String.Optional.* A place to log the alerts.

Important: Ensure that Mediator is configured to send event notifications to the destination(s) you specify here.

For details about alerts and transaction logging, see *Administering webMethods Mediator*.

Value	Description
CentraSite	<p>Sends the alerts to the API's Events profile in CentraSite.</p> <p>Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to Solutions > Mediator > Administration > CentraSite Communication). For information about how to configure communication with CentraSite, see <i>Administering webMethods Mediator</i>.</p>
Local Log	<p>Sends the alerts to the server log of the Integration Server on which Mediator is running.</p> <p>Also choose a value in the Log Level field:</p> <ul style="list-style-type: none"> ■ Info: Logs error-level, warning-level, and informational-level alerts. ■ Warn: Logs error-level and warning-level alerts. ■ Error: Logs only error-level alerts. <p>Important: The Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to Settings > Logging > Server Logger).</p>
SNMP	<p>Sends the alerts to CentraSite's SNMP server or a third-party SNMP server.</p> <p>Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to Solutions > Mediator > Administration > Email). For the procedure, see <i>Administering webMethods Mediator</i>.</p>
Email	<p>Sends the alerts to an SMTP email server, which sends them to the email address(es) you specify here. To specify multiple addresses, use the plus button to add rows.</p> <p>Prerequisite: You must configure the SMTP server destination (in the Integration Server</p>

Administrator, go to **Solutions > Mediator > Administration > Email**). For the procedure, see *Administering webMethods Mediator*.

EDA

Mediator can use EDA to log the payloads to a database.

Prerequisite: You must configure the EDA destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > EDA**). For the procedure, see *Administering webMethods Mediator*.

Service Result Cache

This action enables caching of the results of SOAP and REST API invocations based on the caching criteria that you define. If the Service Result Cache action is set for an API, Mediator enables caching for the API when the API is deployed using the information for the action received from CentraSite. You can define the elements for which the API responses are to be cached based on the criteria: HTTP Header, Path, or XPath Expression. You can also limit the values to store in the cache using a whitelist. And, for the elements that are stored in the cache, you can specify other parameters such as Time to Live (TTL) and maximum response payload size.

Note: In the case of REST based APIs, caching is supported only for the results of the HTTP method GET and not for other methods. Service result cache is not supported for a JSON request in a REST based API but is supported for a JSON response.

Caching the results of an API request:

- Increases the throughput of the API call
- Improves the scalability of the API

Note: We recommend the use of caching only for elements that do not require live data and state values.

Input Parameters

Configure
Caching Based
On

Select a caching criteria. Mediator uses this information to determine the request component that is the actual payload based on which the results of the API invocation are cached. The options are:

- **HTTP Header** Uses the HTTP header in the API request. You can use this criteria for REST based APIs that accept payloads only in HTTP format.

	<ul style="list-style-type: none"> ■ Path Uses the entire invocation URL including the query parameter. This option is applicable mainly for REST based API requests that use the URL or path parameter as the payload. ■ XPath Expression Uses the XPath expression in the API request. You can use this criteria for: SOAP based API requests whose payload is a SOAP envelope, and REST based API requests that accept payloads in XML format.
Header Name	(Only if you select HTTP Header) Specifies the HTTP header name.
Namespace	<p><i>Optional.</i> (Only if you select XPath Expression) Specifies the namespace of the XPath expression:</p> <ul style="list-style-type: none"> ■ Prefix - The prefix for the namespace. For example, soapenv or axis ■ URI - The namespace URI - For example, http://schemas.xmlsoap.org/soap/envelope/ or http://ws.apache.org/axis
XPath Expression	(Only if you select XPath Expression) Specifies the XPath expression in the API request.
Add to Whitelist	<i>Optional.</i> (Only if you select HTTP Header or XPath Expression)Mediator caches the API responses only for requests whose cache criteria matches with the one set for the action, and whose criteria evaluation results in any one of the values in this list.
Time to Live	<i>Optional.</i> Specifies the lifespan (Days Hours Minutes, for example: 5d 4h 1m) of the elements in the cache after which the elements are consider out-of-date.
	<p>Note: The maximum value that you can set for the Time to Live (TTL) parameter is 24855 days (unlimited). If no value is specified, the TTL is unlimited (does not expire). If you set the TTL value to 0d 0h 0m, the API results are not cached.</p>
Maximum Response Payload Size	Specifies the maximum payload size for the API in kilo bytes. The value -1 stands for unlimited payload size.

Mediator uses the Ehcache capability provided by Integration Server to cache the results of the API calls. You can configure caching for a single Mediator node or for a cluster.

For details on configuring Ehcache, see *webMethods Integration Server Administrator's Guide*.

Recommendations and Best Practices for Service Result Caching

This section provides guidance on the use of caching for the results of an API request.

- Caching is not recommended in cases where consumers absolutely rely on current information retrieved from a back-end service.
- As caching will occur in memory, ensure that you are not operating in a memory constrained environment.
- Balance the memory consumption with the API response sizes. If your API returns huge responses or large binary attachments, limit caching by specifying the maximum size of the cache or by defining data eviction policies to avoid excessive memory consumption.
- Design the APIs for which you want to implement caching to be idempotent to support reliable caching.
- Configure a suitable value for the Time to Live (TTL) parameter for the cache entries based on your business needs and the use cases for your API.

For example, if your API serves static product catalog data which is only updated once a quarter choose a large TTL value. If your API serves for example environmental data where a certain age of the data is tolerated, choose a TTL value like 15 minutes.

Note: A TTL value that you set in a standalone Mediator is reset if the Mediator restarts. For example, if you set 15 minutes as the TTL value of an element which is inserted after 10 minutes, and Mediator restarts. The element has a TTL value of 15 minutes after the Mediator is restarted.

- Caching of the results of an API call is recommended:
 - If the response time of a direct query to the database is high or subject to high latencies, and if the data requested does not change frequently or is static.
 - If the client applications can use slightly outdated, cached data. For example, a weather API can be supplied data from a cache which is an hour old.
 - If there are temporary service interruptions on the server side, to mitigate these interruptions with data from the cache.
 - If the API that requests data is subject to traffic management rules, usage quotas, or if billing is based on the number of calls to the API, you can overcome these by using cached data.
- Caching may not be effective or is not recommended:
 - If the response of a direct request for data to the database is very fast and scaling is not an issue, using cached data may not provide additional benefits.
 - If your API uses non-idempotent requests.

Validate Schema

This action validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Mediator can enforce this action for messages sent between APIs. When this action is configured for a proxy API, Mediator validates XML request messages, response messages, or both, against the XML schema referenced in the WSDL.

Input Parameters

Validate SOAP Message(s) *Object.* Validates request and/or response messages. You may select both Request and Response.

<u>Value</u>	<u>Description</u>
Request	Validate all requests.
Response	Validate all responses.

Important: Be aware that Mediator does not remove `wsu:Id` attributes that may have been added to a request by a client as a result of security operations against request elements (that is, signatures and encryptions). In this case, to avoid schema validation failures you would have to add a [Request Transformation](#) action or a [Response Transformation](#) action to the API so that the requests and responses are passed to an XSL transformation file that removes the `wsu:Id` attribute.

Computed Runtime Actions

CentraSite Business UI offers you the possibility to add computed runtime actions into the policy workflow; this gives you the option to define your own runtime action; which means that you can implement your own algorithms for representing the action's user interface.

Computed runtime actions let you create your own layout by using a UI Rendering Concept. You can also specify your own rendering logic to display the computed values. You could, for example, create a custom display of the attribute as a drop down or a radio button.

A computed runtime action can be implemented using the GWT framework. For a computed runtime action, you create an archive file that contains the plug-in definition, and you load the archive file in the CentraSite BUIExtension folder.

Writing Your Own Computed Runtime Action

A computed runtime action can be implemented as a plug-in. The prepared plug-in is a collection of files in a specific directory structure. After implementing the plug-in, the files are copied into the `CentraSiteBUIExtension` folder in `<CentraSiteInstallDir> \demos` directory.

In the following sections, we demonstrate a sample framework named “MyComputedRuntimeAction” that illustrates how a custom computed runtime action may be set up.

You may use this sample as a guideline, adapting it and renaming it to suit your individual requirements. The sample indicates where customization is required.

The Build Environment

This section explains the build environment for generating the files that are used for the GUI and for compiling the necessary Java source files. It assumes the use of Ant, the Java-based build tool.

The following file system structure under the computed runtime action directory is assumed:

Name of File or Folder	Description
src	This folder that holds the Java source files.
lib	This folder contains the archive file, plug-in's executor class and the external libraries.
build.xml	The Ant input file for building the destination files

The Ant file, `build.xml` can be used to establish a custom computed profile.

The classpath for the build step must refer to all JAR files contained in the `redist` folder of the `CentraSite` installation. Add these JAR files to the build path of your java project also.

Implementation Guidelines for Computed Runtime Action

In order to create, install and use plug-ins, you must perform the following tasks:

- [Implementation for Computed Action UI](#)
- [Implementation for Computed Action Parser](#)

This section does not explain all the details of the Java source file; its purpose is to indicate the code that must be modified to suit your environment.

Implementation for Computed Action UI**src\com\softwareag\centrasite\builextension\client\runtime\action
\MyComputedRuntimeActionWidget.java**

```

public class MyComputedRuntimeActionWidget extends Composite {
    private PolicyActionJso policyActionJso = null;
    private TextBox valueBox = null;
    private static final String WARNING_CSS = "loginTextBoxErrorBorder";

    public MyComputedRuntimeActionWidget(String policyActionJson) {
        FlowPanel container = new FlowPanel();
        initWidget(container);

        policyActionJso = getPolicyActionJso(policyActionJson);
        if (policyActionJso == null) {
            Label helloLabel = new Label("The JSON content is empty");
            container.add(helloLabel);
            return;
        }

        //Render widgets
        container.add(getParametersView(policyActionJso));
    }

    private Widget getParametersView(PolicyActionJso policyActionJso) {
        FlowPanel parametersContainer = new FlowPanel();

        JsArray<ParameterJso> parameters = policyActionJso.getParameters();
        if (parameters == null) {
            return parametersContainer;
        }

        for (int i = 0; i < parameters.length(); i++) {
            parametersContainer.add(getParameterView(parameters.get(i)));
        }

        return parametersContainer;
    }

    private Widget getParameterView(ParameterJso parameterJso) {
        FlowPanel parameterContainer = new FlowPanel();
        Label nameLabel = new Label(parameterJso.getName());
        parameterContainer.add(nameLabel);

        valueBox = new TextBox();
        valueBox.setLayoutData(parameterJso.getId());

        String[] values = parameterJso.getValues();
        if (values != null && values.length > 0) {
            valueBox.setValue(values[0]);
        }

        parameterContainer.add(valueBox);
        return parameterContainer;
    }

    public static native PolicyActionJso getPolicyActionJso(String json) /*- {
        return eval('(' + json + ')');
    }-*/;

    public String getJson() {
        JsArray<ParameterJso> parameters = policyActionJso.getParameters();
        ParameterJso parameterJso = null;
        if (parameters != null && parameters.length() > 0) {

```

```

        parameterJso = parameters.get(0);

        String[] values = {valueBox.getValue()};
        parameterJso.setValues(values);
    }

    return policyActionJso.toJSON();
}
public boolean isValid() {
    String value = valueBox.getValue();
    boolean isValid = (value != null && !"".equals(value));
    if (!isValid) {
        valueBox.addStyleName(WARNING_CSS);
    } else {
        valueBox.removeStyleName(WARNING_CSS);
    }

    return isValid;
}
}

```

The `MyComputedRuntimeActionWidget` class extends the class `Composite`, which declares the basic rendering methods for the CentraSite Business user interface.

Implementations and Description

`MyComputedRuntimeActionWidget(String policyActionJson)`

Constructor dictates the user-defined rendering of the action's UI.

`getPolicyActionJson`

Returns the JSON object from the specified object.

`String getJson()`

Returns a JSON encoded string representing the action's parameters.

`boolean isValid()`

Enforces validation logic for the action's parameter values.

Implementation for Computed Action Parser

To implement your own computed runtime action with custom UI rendering, the parser (`MyComputedRuntimeActionParser.java`) must be located in the service directory. A parser is responsible for generating compressed JSON data from the given policy action instance, and creating a custom rendering of the action instance using the JSON data.

Here is the frame of the computed runtime action parser implementation:

src\com\softwareag\centrasite\buil\extension\service\MyComputedRuntimeActionParser.java

```

public class MyComputedRuntimeActionParser extends BasePolicyActionExtensionParser {
    public MyComputedRuntimeActionParser(CentraSiteSession centraSiteSession,
        CentraSitePolicyActionTemplate actionTemplate,

```

```

        CentraSitePolicyActionInstance actionInstance) {
            super(centraSiteSession, actionTemplate, actionInstance);
        }
        @Override
        public CentraSitePolicyActionInstance getActionInstance(String json)
            throws CLLEException {
            Gson gson = new Gson();
            MyComputedRuntimeActionInfo = gson.fromJson(json,
                MyComputedRuntimeActionInfo.class);

            if (actionInfo == null) {
                return null;
            }

            CentraSitePolicyActionInstance policyActionInstance = null;
            CentraSiteObjectManager objectManager =
                getCentraSiteSession().getCentraSiteObjectManager();

            if (actionInfo.isActionInstance()) {
                policyActionInstance = objectManager.getPolicyActionInstance(action
Info.getId());
            } else {
                policyActionInstance = objectManager.createPolicyActionInstance(act
ionInfo.getId());
            }

            if (policyActionInstance == null) {
                return null;
            }

            setParameterValues(policyActionInstance, actionInfo);
            return policyActionInstance;
        }
        private void setParameterValues(CentraSitePolicyActionInstance
            policyActionInstance, MyComputedRuntimeActionInfo actionInfo)
            throws CLLEException {
            List<MyComputedRuntimeParameterInfo> parameters = actionInfo.getParame
ters();

            if (parameters == null || parameters.isEmpty()) {
                return;
            }

            MyComputedRuntimeParameterInfo parameterInfo = parameters.get(0);
            Collection<Object> convertedParameterValues = new ArrayList<Object>();
            convertedParameterValues.addAll(parameterInfo.getValues());
            policyActionInstance.setAttributeValue(parameterInfo.getId(),
                convertedParameterValues);
        }
        @Override
        public String getJson() throws CLLEException {
            Gson gson = new Gson();

            MyComputedRuntimeActionInfo actionInfo = null;
            if (getActionInstance() != null) {
                CentraSitePolicyActionTemplate policyActionTemplate =
                    getActionInstance().getCentraSitePolicyActionTemplate();
                actionInfo = new MyComputedRuntimeActionInfo(getActionInstance().g
etId(),
                    policyActionTemplate.getName());
                actionInfo.setActionId(policyActionTemplate.getId());
                actionInfo.setIsActionInstance(true);
            } else if (getActionTemplate() != null) {
                actionInfo = new MyComputedRuntimeActionInfo(getActionTemplate().g
etId(),

```

```

        getActionTemplate().getName());
        actionInfo.setActionId(getActionTemplate().getId());
    }

    fillParameterInfos(getActionTemplate(), actionInfo);
    return (actionInfo != null ? gson.toJson(actionInfo) : null);
}
private void fillParameterInfos(CentraSitePolicyActionTemplate actionTemplate,
    MyComputedRuntimeActionInfo actionInfo) throws CLLEException {
    if (actionTemplate == null) {
        return;
    }

    Collection<CentraSiteObjectAttribute> attributes = actionTemplate.getAttributes();
    if (attributes == null || attributes.isEmpty()) {
        return;
    }

    List<MyComputedRuntimeParameterInfo> parameters = new
    ArrayList<MyComputedRuntimeParameterInfo>(attributes.size());
    MyComputedRuntimeParameterInfo parameter = null;
    for (CentraSiteObjectAttribute attribute : attributes) {
        parameter = new MyComputedRuntimeParameterInfo(attribute.getName(),
            attribute.getDisplayName());
        parameters.add(parameter);
    }

    actionInfo.setParameters(parameters);
}
}

```

Setting up the Computed Action Plug-in

The following list shows the main methods on each of the two Java source files `MyComputedRuntimeActionWidget.java` and `MyComputedRuntimeActionParser.java` and describes the type of functions that they serve.

- The `getPolicyActionJson` method returns the JavaScript Object (JSO) from the given JSON-formatted string.
- The `getActionInstance` method returns a `CentraSitePolicyActionInstance` object from the given JSON-formatted string.
- The `String getJson` method returns a JSON-formatted string from the existing policy action instance.
- The boolean `isValid()` method enforces a validation logic for the user-defined rendering of the runtime action.

Assuming that you have set up all the Java files correctly in the directories, you should be able to build with the command:

```
ant -f build.xml jar all
```

Activating the Computed Action

After you define the computed action as a plug-in (extension point) with the above steps, enable the computed action in the Business UI configuration file `centrasite.xml` in order to display the action in the policy accordion.

Important: Remember that the action parameters defined in the configuration file are editable and cannot be protected.

To activate the plug-in

1. Open the `centrasite.xml` file.
The configuration file is located in the `cast\cswebapps\BusinessUI\system\conf` directory.
2. Navigate to the property lines `<UIProperties>` -> `<Extensions>` -> `<PolicyActions>`
3. Append the property statement for your custom computed runtime action (MyComputedRuntimeAction) as below:

```
<PolicyActions>
  <PolicyAction id="uddi:44e3e2de-064c-432f-b67a-8fbca0fb04d6"
    class="com.softwareag.centrasite.bui.extension.service.
MyComputedRuntimeActionParser" />
</PolicyActions>
```

wherein,

Parameter	Description
<code>id</code>	A unique identifier for the computed action. It uniquely distinguishes an action in the CentraSite registry. If you wish to reconfigure the action at a later stage, you identify the action using this id.
<code>class</code>	A parser implementation for the computed action.

4. Save and close the configuration file.
5. Restart Software AG Runtime.

Sample Computed Runtime Action

Your CentraSite installation contains a sample computed runtime action (which is contained in `demos` folder) that you can use to create an archive file for the custom runtime action specific to the CentraSite Business UI.

- `SampleComputedRuntimeAction`