Administering webMethods Mediator

Version 9.7

October 2014

This document applies to webMethods Mediator Version 9.7 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.

Document ID: SMG-AG-97-20141015

Table of Contents

About this Guide

This guide is for administrators of webMethods Mediator. It provides an overview of how Mediator operates and explains administrative tasks such as connecting Mediator to CentraSite and configuring security and logging. It also explains concepts such as clustering, load balancing, architecture and monitoring.

Document Conventions

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

Documentation Installation

You can download the product documentation using the Software AG Installer. The documentation is downloaded to a central directory named _documentation in the main installation directory (SoftwareAG by default).

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 and certified samples, 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, 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 Introduction

What Is 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 can also be used with Web services that have been externalized as Application Programming Interfaces (APIs).

Mediator is designed for use with Software AG Service-Oriented Architecture (SOA) products. The Mediator application is delivered as a package called WmMediator, which runs on Integration Server. It provides an infrastructure for the run-time enforcement of service policies that are defined and managed from Software AG's UDDI registry/repository, CentraSite.

Mediator serves two primary functions:

- To serve as an intermediary between service consumers and service providers (*mediation)*.
- To serve as a *policy enforcement point (PEP)* that enforces policies defined for a Web service.

Mediation and Policy Enforcement

Through service virtualization, Mediator serves as an intermediary between service consumers and service providers. This means that service requests sent from a service consumer actually go to a virtual service hosted on Mediator for processing rather than directly to the service provider. A *virtual service* is an enhanced copy of a service provider's Web service, and acts as the consumer-facing proxy for the provider's Web service. You configure virtual services in CentraSite and deploy them to a Mediator server.

Alternatively, if you have Web services that have been externalized as APIs, the requests sent from a service consumer go to a virtualized API hosted on Mediator for processing. You virtualize an API using the CentraSite Business UI, and deploy them to a Mediator server.

You can create policies for virtual services or virtualized APIs, which provide run-time governance capabilities for them. A *policy* is a sequence of actions that is carried out by Mediator when a consumer requests a particular service through Mediator. The actions in a policy perform activities such as identifying/authenticating consumers, validating digital signatures and capturing performance measurements. An *action* is a single task that is included in a policy and is evaluated by Mediator at run time. Actions have one or more parameters, which you configure when you insert the actions into a policy. For example, an action that identifies consumers specifies one or more identifiers to identify the consumers who are trying to access the services.

Mediator provides built-in action templates. A built-in action template is a definition of an action that can be used in a policy. An action template specifies the set of parameters

associated with a particular policy action. You can use these action templates to create actions for your policies.

Mediation also provides improved interoperability between consumers and providers. Since all requests from the service consumer pass through Mediator, you can configure the virtual service to make any necessary changes to the message or its protocols before engaging with the provider.

Example

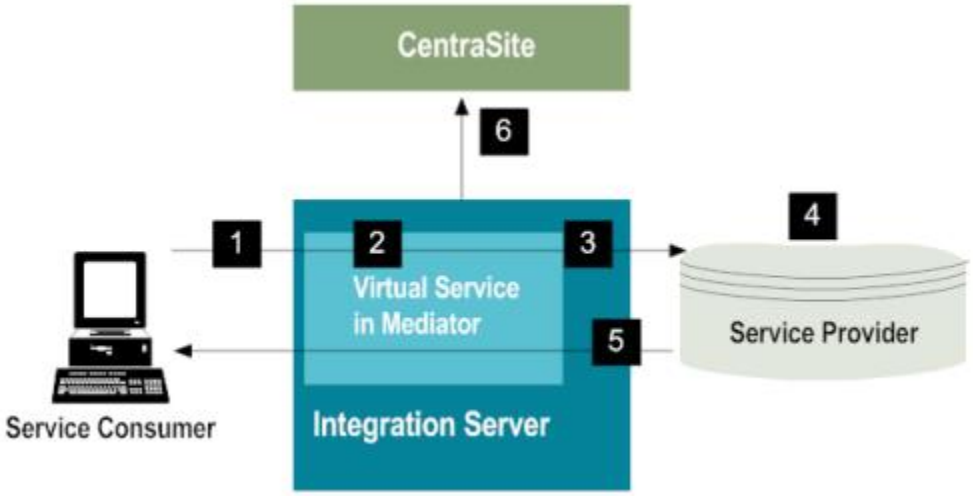
In a regular scenario, a request from a consumer application goes directly to a Web service exposed by the service provider. In a mediated system, the request goes through Mediator, where policies are applied to the Web service and are enforced by Mediator.

For example, a consumer application could be a Web form filled out by a bank employee looking for all the records for a customer: savings, investment, and mortgage. The Customer Data service handles requests for such data, so Mediator applies the policies to the request. The policies it enforces for the service dictate that the employee making the request is eligible to see the mortgage data, but none of the other data requested. The service retrieves the mortgage data from the Financial Holdings service provider and returns it to the consumer application.

Mediator in the SOA Landscape

Mediator is built on the same run-time platform as Integration Server, and provides the enforcement of service policies that you define and manage in CentraSite. To perform this role, Mediator depends on virtual services, which you also create and manage in CentraSite, that are deployed to Mediator through the use of *virtual service definitions (VSDs)*. Once the virtual service is deployed, Mediator performs run-time policy enforcement on the virtual service. When Mediator receives requests, it forwards them to the service provider (if the request satisfies the service policy) and returns the response to the consumer.

The following diagram shows the run-time interactions of Mediator in the system.



Step	Description
1	A service consumer makes a call to a Web service running on a service provider.
2	The virtual service running on Mediator receives and processes the message.
3	The message is sent to the service provider.
4	The service provider retrieves the required data or performs the proper task.
5	The service provider, through the native service, sends the requested data back through Mediator to the consumer application.
6	Throughout the process, Mediator sends monitoring data through SNMP traps to CentraSite.

Mediator Key Capabilities

Mediator provides several important capabilities to your SOA system.

Service Virtualization

The virtualization of Web services provides the following capabilities.

Loose Coupling Between Consumers and Providers

Loose coupling between service consumers and service providers enables greater flexibility in your SOA system. Loose coupling provides independence of location, protocol, and format between the consumers and providers. This means that changes made by either a consumer or a provider do not necessarily require a change in the other. This is because there is no direct connection between the two; instead, they depend upon Mediator for the connection. The Web service that the consumer invokes is hidden from the consumer application, with Mediator serving as the mediation layer.

Because virtual services are created on CentraSite and deployed to Mediator, communication between the two is vital. CentraSite must communicate any virtual service updates to Mediator. CentraSite uses the target endpoint you register when you specify the target (Mediator) to automatically communicate changes in the virtual services to the instance of Mediator to which it is deployed.

Similarly, Mediator relies on CentraSite for any updates to the consumer applications that are authorized to access the virtual services.

Monitoring of Run-Time Events and Performance Metrics

Mediator enables you to report run-time events and performance metrics to CentraSite for all virtual services that are deployed to Mediator. You can view these events and performance metrics in CentraSite, and configure your policies to send alerts when some kinds of events occur. For more information, see "Key Performance Indicator Metrics and Run-Time Event Notifications" on page 21.

Message Transformation, Pre-Processing and Post-Processing

You can optionally configure a virtual service if you need to transform the service request and response messages to suit your special needs. To do this, you can specify an XSLT file to transform the messages during the mediation process.

In addition, you can optionally configure virtual services to invoke Integration Server services to pre-process or post-process the request/response messages. All procedures for creating virtual services appear in the CentraSite documentation.

Policy Enforcement

The policies you create for virtual services provide run-time governance capabilities for the virtual services. Similarly, the policy enforcement rules you create for virtualized APIs provide run-time governance capabilities for the APIs. Mediator provides built-in run-time actions that you can include in a policy or policy enforcement rule, and then configure their parameters to suit your needs. These actions perform activities such as identifying/authenticating consumers, validating digital signatures and capturing performance measurements.

There are two separate sets of built-in run-time actions you can use:

- Run-time actions for virtual services.

  You use these actions only when you are using CentraSite Control to create run-time policies for virtual services.

- 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.

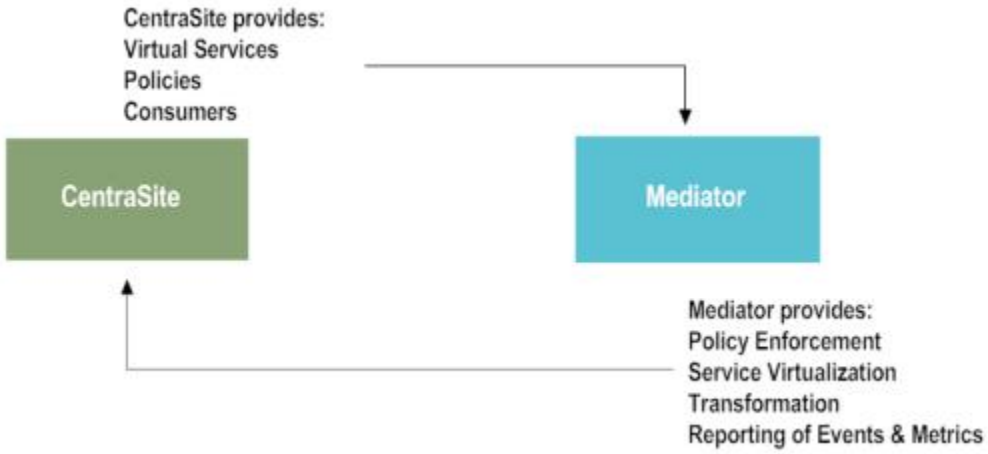
For more information, see "The Built-In Run-Time Actions" on page 43.

Consumer Provisioning

Mediator maintains a list of consumer applications specified in CentraSite that are authorized to access the virtual services deployed to Mediator. Mediator synchronizes this list of consumer applications through a manual process initiated from CentraSite.

Seamless Integration with CentraSite

Mediator hosts the virtual services that you create in CentraSite. Part of defining virtual services is configuring the policies that Mediator will enforce and the consumer applications that Mediator will monitor. When you deploy virtual services to Mediator, and service consumers make requests of the service provider, Mediator mediates and enforces the policies defined for the virtual service. Mediator then reports the collected events and performance metrics to CentraSite.



Clustering Support

Multiple instances of Mediator can be clustered together to provide scalability of mediation. A load balancer can be placed in front of the clustered Mediator instances to properly distribute the request messages. For more information about load balancing, see "Clustering and Load Balancing" on page 29.

The Components That Mediator Uses

Mediator requires inputs from the following components in the system to process requests and to enforce policies.

- Service providers.
- Service consumers.
- CentraSite.
- The Mediator configuration options in Integration Server.
- Virtual services.
- Virtualized APIs.

Service Providers

Service providers are the applications and systems that provide the services that are exposed as Web services in your SOA architecture. They can consist of legacy systems, ERP systems, CRM systems or other systems. The services they provide can be leveraged throughout the SOA system by service consumers.

Service Consumers

A *service consumer* is an application, system, or technology that makes Web service calls on a native service for the data and tools necessary to complete a specific task. An example of a service consumer could be a Web form that is used to retrieve customer data from a service provider. Consumer applications are one type of service consumer that are kept synchronized with Mediator.

CentraSite

CentraSite provides the design-time environment that you will use to create or reference all the assets of your SOA-based application, including virtual services, policies, consumers, XML schemas, BPEL processes, and more. These assets, as well as the service providers' Web services, are published to CentraSite's UDDI registry/repository. The Web services can be imported from many places, including Integration Server.

In addition, you use CentraSite to view run-time events and performance metrics.

For details, see the CentraSite documentation.

The Mediator Configuration Options in Integration Server

You need to configure some or all of the following Mediator configuration options in Integration Server Administrator:

- The communication parameters required for Mediator to exchange data with CentraSite.
- The SNMP destination for sending run-time events. Mediator uses SNMP traps to capture events, which you can send to either the CentraSite SNMP server or a third-party SNMP server.
- The email destinations for sending monitoring alerts or for logging transaction payloads.
- The endpoints to use for load balancing routing, if you want to distribute requests among multiple endpoints.
- The keystores and truststores that are required for message-level security. They provide SSL authentication, encryption/decryption, and digital signing/verification services for all message content that Mediator sends.
- The HTTP or HTTPS ports on which Mediator and the deployed virtual service will be available.
- The service fault responses that are returned to consuming applications.
- You can configure Mediator to act as a Security Token Service (STS) client.
- And more.

For details, see the chapter "Configuring Mediator " on page 53.

Virtual Services

A *virtual service* is an enhanced copy of a service provider's Web service, and acts as the consumer-facing proxy for the provider's Web service. You configure virtual services in CentraSite and deploy them to a Mediator server.

At design time, you virtualize a Web service (make a copy of it) and then specify additional metadata that defines:

- A *target type*. A target type is the type of server/PEP application to which the virtual service will be deployed. Mediator is a target type.
- A *target*. A target is an object that represents a specific instance of a target type; for example, a specific Integration Server that hosts Mediator.
- The consumer applications that will be allowed to access the virtual service.
- The transport protocol that the consumer application must use to communicate with the virtual service.

- The virtual service's routing protocol, which specifies how the virtual service will route the service requests to the native service endpoint. For HTTP or HTTPS requests, you can choose the following routing protocols:
  - "Straight Through" routing (to route requests directly to the native service endpoint).
  - "Context-Based" routing (to route specific types of messages to specific endpoints according to context-based routing rules).
  - "Content-Based" routing (to route specific types of messages to specific endpoints based on specific values that appear in the request message).
  - "Load Balancing" routing (to distribute requests among multiple endpoints).
- Any optional transformation files to transform the request/response messages.
- Any optional Integration Server services to pre-process or post-process the request/response messages.
- The policy or policies for the virtual service.

For details, see the CentraSite documentation.

Virtual Service Definitions

When you deploy the virtual service to a Mediator server, Mediator generates an XML document called a *virtual service definition (VSD)*. The VSD defines the virtual service for Mediator, and contains all the resources required to deploy the virtual service to a Mediator server, including the policy that applies to the service. You cannot edit the VSD.

Virtual Service Synchronization

When a virtual service is created and published to the CentraSite UDDI registry/repository, its WSDL contains no concrete endpoint until it is deployed to at least one instance of Mediator. After the virtual service is deployed to Mediator, Mediator updates its WSDL with an endpoint that points to the virtual service's location on Mediator. Then the virtual service location is updated in CentraSite. This action synchronizes CentraSite and Mediator.

After the virtual service WSDL is deployed from CentraSite to Mediator, and the WSDL with the new endpoint is redeployed to CentraSite, Mediator is aware of the endpoint of the virtual service, what policies to enforce, routing and load balancing requirements for the virtual service, and what event and performance data to send back to CentraSite. This event and performance data can be monitored in the CentraSite monitoring interface.

Additionally, CentraSite knows the new endpoint of the virtual service. Then a consumer application that is authorized to view the CentraSite UDDI registry/repository and to use the virtual service can locate the virtual service in the registry and invoke it.

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.

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 distribution channel, CentraSite 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, and 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 details, see the CentraSite documentation sections *Virtualizng APIs Using the CentraSite Business UI* and *Run-Time Governance Reference*.

2 Key Performance Indicator Metrics and Run-Time Event Notifications

Overview

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 target. 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 target.

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 following sections describe:

- The predefined run-time events and metrics.
- The event notification destinations to which you can publish the events and metrics.
- Sending alerts and logging transactions.
- The metrics tracking interval.

To enable Mediator to publish metrics and events, be sure to set the configuration options as described in "Configuring Communication with CentraSite " on page 56.

In addition, you must configure CentraSite to receive run-time events and metrics, as described in the section *Run-Time Governance* in the CentraSite documentation.

The Run-Time Events

The types of run-time events that Mediator can publish are as follows:

Event Type	Description
Lifecycle	A Lifecyle 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.
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). Transaction events are generated by the run-time action Log Invocations.
Monitoring	Mediator publishes key performance indicator (KPI) metrics, described below. 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.

The Key Performance Indicator (KPI) Metrics

For the Monitoring event type, Mediator can publish the following types of KPI metrics:

Metric	Reports...
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.

Metric	Reports...
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.
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 `pg.PgMetricsFormatter.includeFaults` parameter to true. For more information, see "Advanced Settings" on page 121.

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 "SNMP Destinations for Run-Time Events" on page 60.
  - 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, "SNMP Destinations for Run-Time Events" on page 60.
- An EDA destination. Mediator can use EDA to publish run-time events and metrics to the following:
  - Database: Mediator uses a JDBC connection pool that you need to define for use by Mediator in the Integration Server.
  - Messaging server such as webMethods Universal Messaging: If you choose the EDA destination as the default EDA endpoint, Mediator publishes the events and

KPIs to the messaging server which then sends the same to the EDA default JMS endpoint.

For the procedure to configure Mediator to send this data to an EDA destination, see "EDA Configuration for Publishing Run-Time Events and Metrics" on page 57.

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.
- 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.

The SMTP Email Server Destination

To specify an SMTP email server 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 "SMTP Destinations for Alerts and Transaction Logging" on page 67.

The Integration Server Local Log Destination

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 built-in 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 Audit Log Destination

The Integration Server Audit Log destination is available only for the Log Invocation action. 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 the *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 (see "Configuring Communication with CentraSite " on page 56).

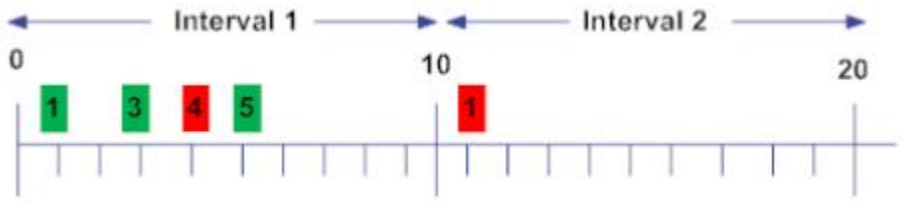
Mediator only tracks metrics for the current interval. At the end of the interval, Mediator aggregates the metrics and reports them to CentraSite. Once the metrics are reported, Mediator resets its counters for the new interval. Mediator does not calculate and aggregate metrics across intervals. If Mediator 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.

Examples of interval metric publishing

For example, suppose that the tracking interval is 10 minutes. One of the key performance indicator (KPI) metrics is "Availability", which reports the amount of time that a service was available during the current interval, shown as a percentage. The green boxes below indicate successful requests and the red ones indicate unsuccessful requests.

A request is considered unsuccessful when a network fault occurs or when the back-end service is unavailable for any reason. In the case of a normal application-level SOAP fault, Mediator considers the request to be successful. In the illustration below, in Interval 1 (0 - 10 minutes) a request failed at the 4 minute mark, followed by a successful request at the 5 minute mark. Therefore, Mediator considers the interval between 4 and 5 minutes to be service downtime (even though this may not be accurate). So in this case, for Interval 1 the availability is 9/10 (90%). In the case of Interval 2, only one request was sent, and it failed at the 1 minute mark. Therefore, Mediator considers the time between 1 minute to the end of the interval as service downtime. So the time between the start of Interval 2 (the 10 minute mark) to the failed request is service uptime (1 minute); the availability is 1/10 (10%) for Interval 2. At the end of the interval, Mediator resets the KPI metrics.



3 Clustering and Load Balancing

Overview

Mediator supports clustering to achieve load balancing. In a load balanced system, calls from service consumers and events and metrics (*messages*) are distributed among two or more different instances of Mediator, referred to as *nodes*. Load balancing provides the following benefits:

- **Scaling.** Scaling is provided by distributing the processing of messages across two or more different nodes.
- **Reliability.** A cluster provides fault tolerance, which ensures that if a node goes down, run-time events and metrics can be recovered. For more information, see "Role of the Shared Cache in Metrics and Event Notification" on page 36.

Clustering relies on the clustering feature provided by Integration Server. Each node runs on a separate instance of Integration Server, and so you must configure clustering properly in Integration Server in order for Mediator's clustering feature to work properly. For more information, see the *webMethods Integration Server Clustering Guide*.

Nodes and Clusters

Each node is an instance of Mediator running on an instance of Integration Server. When you configure each node of the system to communicate with the same CentraSite server, you create a *cluster*. A cluster is a group of nodes that monitors the same virtual service and sends events and metrics triggered by that virtual service to CentraSite.

Communication in a Cluster

Communication in a cluster is *peer-based*. In a peer-based cluster, any node in the cluster can perform processing tasks, including processing virtual service requests. However, nodes process messages differently depending on the task as follows:

- When CentraSite deploys virtual service and consumer application data to Mediator, the load balancer selects the processing node.
- When Mediator processes service calls, the load balancer determines which node processes the service call.
- When Mediator sends event notifications and metrics to CentraSite, the node that is the first available to process the message does so.

Nodes can perform any task required by messages they receive from CentraSite.

Note: If communication between the cluster and CentraSite is disabled, then the cluster cannot report metrics. However, if you configured the cluster to report run-time event notifications to CentraSite over SNMP, these event notifications will continue if the CentraSite SNMP destination is enabled and configured correctly. For more information

about the role of the shared cache for deployment, see "Role of the Shared Cache in Deployment" on page 33. For more information about the role of the shared cache for metric notification, see "Role of the Shared Cache in Metrics and Event Notification" on page 36.

Load Balancers

Clustering in Mediator requires a *load balancer*. The load balancer is a third-party tool that routes incoming virtual service calls from CentraSite to the nodes in the cluster. For Mediator, the load balancer:

- Provides CentraSite with a single point of entry to the cluster. This means that CentraSite recognizes only that it is communicating with a single entity rather than multiple nodes. All communication from CentraSite to the cluster goes through the load balancer. CentraSite deploys virtual services to the load balancer, and whichever node is prepared to take action processes the message.
- Distributes calls from service consumers to the virtual services across the cluster. When a service consumer calls a virtual service, the call is routed by the load balancer to the node prepared to process the call.

Load Balancer URLs

Load balancer URLs define for CentraSite the endpoints of the nodes in a cluster. When a service is deployed to a cluster, the node that performs the deployment sends the load balancer URL as the new endpoint to CentraSite as part of its virtual service status message.

CentraSite stores this load balancer URL endpoint in the registry/repository. Since all of the nodes in the cluster use the same load balancer URL, CentraSite accepts messages from any node in the cluster as if they came from a single instance of Mediator.

A load balancer URL consists of a host name (or IP address) and port number of the load balancer as follows:

`http://hostname:portnumber`

or

`http://IP-address:portnumber`

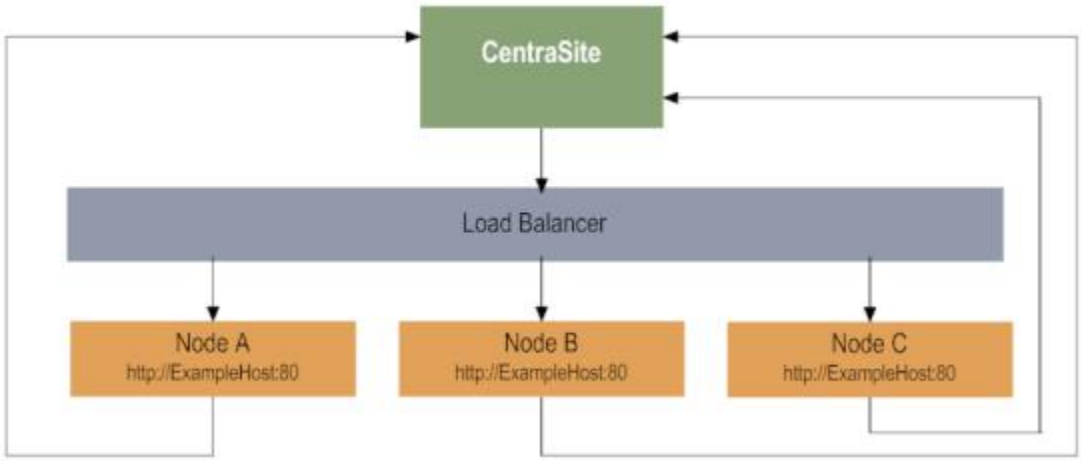
Note: You can also configure the load balancer URL to use the HTTPS protocol.

For example, if the host name of the load balancer is ExampleHost, and its port number is 80, the load balancer URL would be:

`http://ExampleHost:80`

You must configure all the nodes in a cluster with the same load balancer URL. For information about configuring the load balancer URL for Mediator, see "Load Balancing Configuration" on page 69.

The following diagram shows how the load balancer works in the clustered system.



Creating a Mediated Cluster

To create a cluster that includes Mediator, you must configure Integration Servers, a third-party load balancer, CentraSite, and instances of Mediator.

Configuring Integration Server

Mediator's cluster implementation is built upon the Integration Server's cluster support. You must configure the Integration Servers in the cluster as described in the *webMethods Integration Server Clustering Guide*.

Configuring the Third-Party Load Balancer

You must configure a third-party load balancer to use clustering in Mediator. You must use the load balancer to configure the following:

- A virtual network that defines the IP addresses of the nodes
- The WS-Context to route calls to the cluster

For information about configuring your load balancer for use in the clustered system, see the documentation for that product.

Configuring Mediator

All Integration Server cluster members must contain identically-configured instances of Mediator. You use the Mediator Administration console to configure each instance of Mediator. For more information, see "Configuring Mediator " on page 53.

Note: You can use the package replication functionality in the Integration Server Administrator to copy Mediator packages to other servers in the cluster. For information about package replication, see the *webMethods Integration Server Administrator's Guide*.

Configuring CentraSite

When you configure CentraSite, you must configure the deployment endpoint of the target to point to the load balancer. For more information, see the *Run-Time Targets* section of the CentraSite documentation.

Deployment in a Cluster

You can deploy, undeploy and redeploy virtual services and update consumer application information for a cluster the same way you do for a stand-alone instance of Mediator, as described in the sections *Virtualized Services in CentraSite Control* or *Virtualizing APIs Using the CentraSite Business UI* of the CentraSite documentation.

Role of the Shared Cache in Deployment

As part of the deployment process, the cluster synchronizes the nodes with the *shared cache* to ensure that tasks are not processed repeatedly. The shared cache enables clusters to share deployment information and serves as a repository for all virtual service and consumer application information, such as the following:

- Virtual service information and the associated deployment state of the cluster (Initializing or Running).
- Consumer application information using the virtual services deployed to the cluster.

The shared cache resides on each node in the cluster. When any node in the cluster processes a deployment task, the shared cache of the Mediator that processed the task propagates the data to the shared cache of the other Mediator instances in the cluster, keeping all the nodes in the cluster in sync.

Synchronizing Node

Only a single node in the cluster can process a deployment task at any one time. The node that processes this task is referred to as the *synchronizing node*. A node becomes the synchronizing node by being the first to process a task and can be a different node for each task.

When the load balancer receives updates to virtual services and consumer applications from CentraSite, it selects the synchronizing node to process the updates. Once the synchronizing node gets the updates from the load balancer, it updates the shared cache with the changed information. The remaining nodes in the cluster monitor the shared

cache for updates and deploy the virtual services without interacting with either the load balancer or CentraSite.

Initialization of a Cluster

When you start a cluster, you should first start one of the instances of Mediator and allow it to start completely before you start the other instances of Mediator. If you start all instances of Mediator at the same time, that could result in virtual services not getting deployed in some of the cluster nodes.

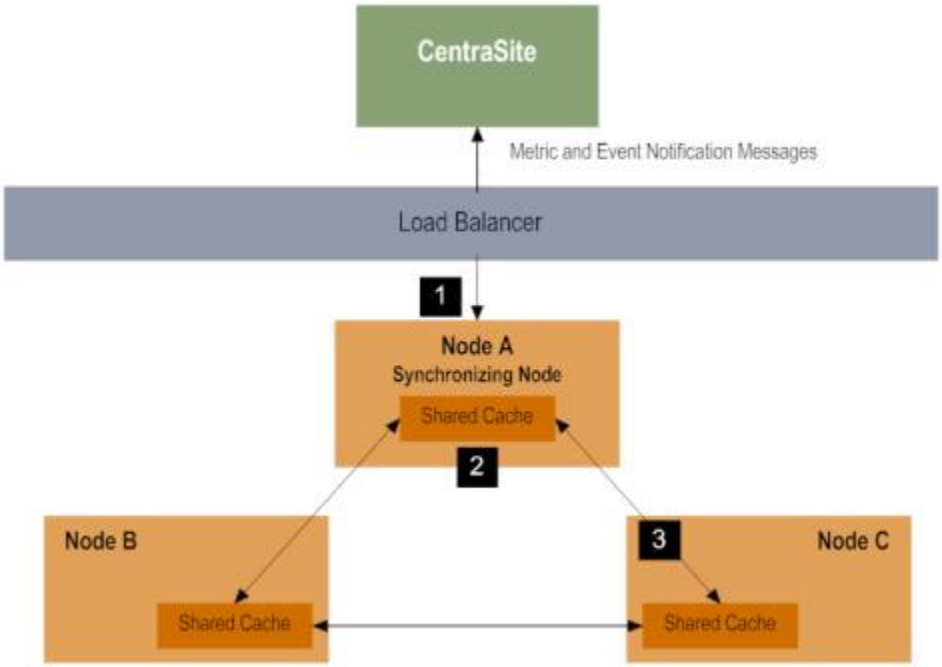
When you start a cluster, it automatically initializes itself as follows:

- The synchronizing node locks the shared cache and sets the cluster's deployment state to Initializing.
- The synchronizing node loads to the shared cache any virtual service definitions and consumer application data that might have been previously stored in that node.
- The other nodes in the cluster retrieve the virtual service definitions and consumer application data from the shared cache (if any). This ensures that every node on the cluster is synchronized with the same information.
- The synchronizing node unlocks the shared cache and changes the cluster's deployment state to Running.

If the initialization was successful, the cluster is now fully operational and ready for CentraSite to deploy virtual service definitions and consumer application data, as described below.

Communication During Deployment

The interaction between nodes during deployment is shown in the following diagram:



Step	Description
1	The synchronizing node receives the virtual service definitions and consumer application data from CentraSite.
2	The synchronizing node: 1. Locks the shared cache and changes the cluster's deployment state from Running to Initializing. 2. Performs the deployment operation. 3. Changes the cluster's deployment state to Running, and releases the cache lock.
3	The other nodes in the cluster retrieve the virtual service definitions and consumer application data from the shared cache. This ensures that every node on the cluster is synchronized with the same information.

Processing Service Requests in a Cluster

Clusters receive service requests from service consumers through the load balancer. When a service consumer makes a call to a virtual service monitored by the cluster, the load balancer receives the message and distributes it to the node that is ready to process

it. For example, if the load balancer is configured to use round-robin distribution, the load balancer distributes each virtual service request to the nodes in turn.

Metric and Event Notification in a Cluster

A cluster collects monitoring and performance data and publishes it to CentraSite similarly to that of a single instance of Mediator. The difference is that a cluster distributes processing across all nodes, thereby balancing the notifications between the nodes. For more information about events and metrics processed by Mediator, see "Key Performance Indicator Metrics and Run-Time Event Notifications" on page 21.

Role of the Shared Cache in Metrics and Event Notification

As part of the metrics and event notification process, the cluster uses the shared cache to:

- Register the policy actions configured in, and received as part of, the virtual service definition deployed from CentraSite. The shared cache provides a cluster-wide view of cached policy actions for the cluster (i.e., the Log Invocation action and the Monitoring actions).
- Store aggregated metrics reported by the nodes in the cluster. Once all of the metrics for a particular service are reported to the shared cache, the data can be consolidated to minimum, maximum, and average response times and success and failure rates. This data is stored in the shared cache until it is time to report the data to CentraSite.
- Provide fault tolerance for the cluster. Normally, any content stored in the memory of the node as a queued task will still be lost if the node goes down. However, if the run-time events or metrics are written to the shared cache, they can be recovered.

Senior Node

Every cluster contains exactly one *senior node* that processes the registered list of policy actions on the shared cache and executes events and aggregated metrics. The senior node is responsible for ensuring that all events and metrics written to the shared cache are processed by a node in the cluster. The senior node runs at 15-second intervals in which it scans the shared cache for events and metrics that are scheduled for execution.

When the senior node scans the list of policies and metrics in the shared cache and determines that tasks are in need of processing, it sends a *processing event* to all the nodes in the cluster. The processing event informs the nodes on the cluster that data must be reported to CentraSite. The first node to respond to the processing event reports the event or metric to CentraSite.

The cluster designates the senior node internally according to which cluster member has been in the cluster the longest. If the senior node becomes disabled, the node left in the cluster that has been part of the cluster the longest takes its place.

Note: The senior node has a different function than the synchronizing node described on "Synchronizing Node" on page 33. The synchronizing node performs a function for deployment, and can be a different node for every transaction. The senior node performs functions for event and metric notifications, and only changes if the node performing the duties of the senior node is removed from the cluster.

Processing Interval

Because nodes send metrics and aggregated events to the shared cache, they are not sent to CentraSite immediately after a Web service is invoked. This is because the senior node only scans the list of events and metrics in the shared cache at a predefined 15-second interval, called a *tick interval*.

In addition, you configure the policy actions and metrics reporting tasks to run at their own particular interval (in minutes) as follows:

- You set the *alert interval* for the Monitoring policy actions in a virtual service, to specify the time interval at which to issue alerts. For more information, see "Key Performance Indicator Metrics and Run-Time Event Notifications" on page 21.
- You set the *publish interval* for metrics, to specify how often Mediator should report performance metrics data. You set the publishing interval in the Mediator Administration screens. For more information, see "Configuring Communication with CentraSite " on page 56.

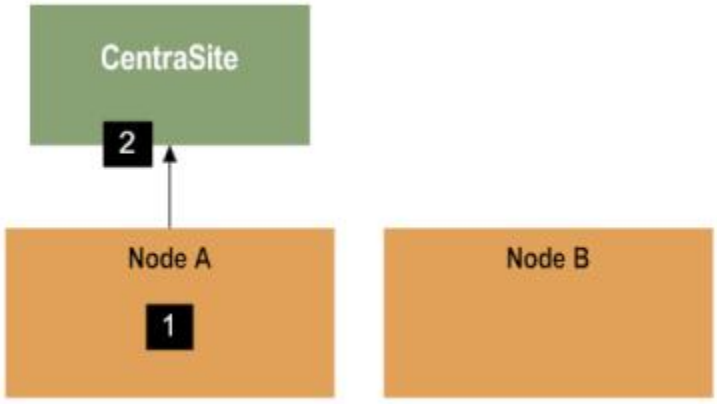
While the tick interval determines how often the senior node scans the list of policy actions and metrics in the stored cache, the policy actions and metrics cannot be processed until the time interval for each has been met. For example, if the policy interval for metrics reporting is configured to run every 10 minutes, then the senior node processes metrics every 40 tick intervals. This is because there are 4 tick intervals every minute for 10 minutes: so it takes 40 tick intervals before the 10-minute policy interval for the metrics is reached (4 * 15 seconds * 10 = 10 minutes). At that time, the senior node can send a processing event to the nodes in the cluster, and the responding node reports the metrics to CentraSite.

Reporting Non-Aggregated Run-Time Events

All non-aggregated run-time events are processed by the node that mediated the Web service invocation. In this situation, the node that triggers the event sends the event notification directly to CentraSite. Neither the senior node, nor any other nodes in the cluster, are involved in the event notification.

Non-aggregated events include:

- Error events
- Policy Violation events
- Lifecycle events

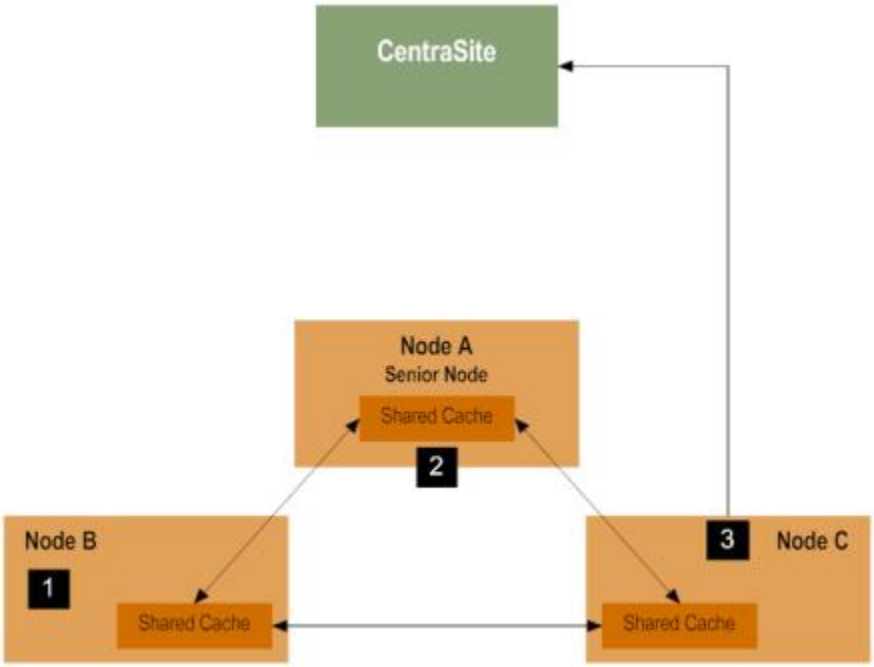


Step	Description
1	The node produces an event.
2	The node reports the event to CentraSite.

Reporting Aggregated Events and Performance Data in a Cluster

When aggregated events or performance data metrics are collected by a node in a cluster, that node sends the event or metric to its shared cache where it awaits processing.

At each interval, the senior node scans the shared cache for the events and metrics stored there. The senior node then notifies all the nodes in the cluster of the stored metrics and events. The first node to respond to the senior node's notification processes the event or metrics, reporting them to CentraSite and the other nodes in the cluster will disregard the notification. Any of the nodes in the cluster are eligible to send the metrics to CentraSite.



Step	Description
1	The aggregated events or metrics are produced by any or all nodes in the cluster, and each node that collected the events and metrics publishes them to its own shared cache.
2	At each tick interval, the senior node scans the events and metrics in the shared cache. If the interval for processing the metrics or events has been met, the senior node notifies all the nodes in the cluster that the events or metrics are ready to be processed.
3	The first node to respond to the senior node's notification processes the event or metrics and reports them to CentraSite. Any of the nodes in the cluster are eligible to send the metrics to CentraSite.

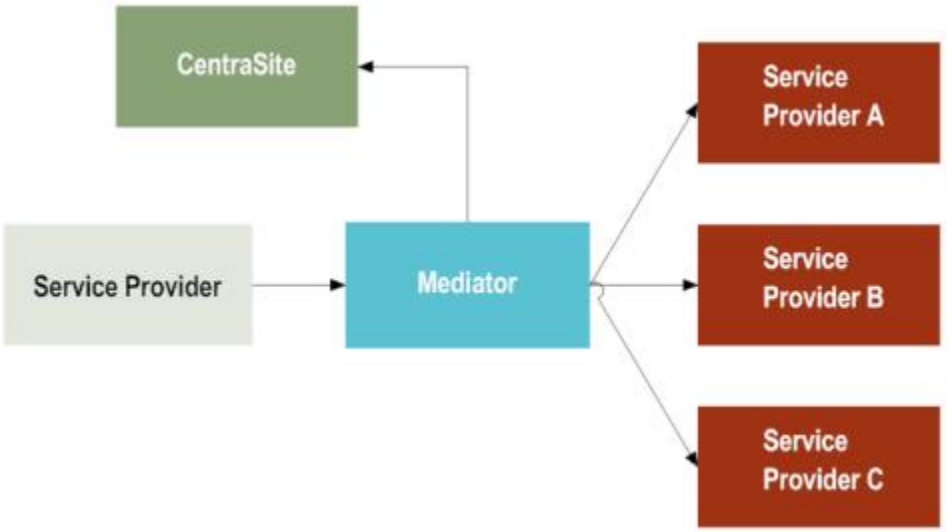
Load Balancing Service Providers

You can configure a virtual service in CentraSite to configure a virtual service to load balance requests between several service providers without clustering. In this scenario, you create a virtual service that uses a load balancing routing protocol and deploy it to a single instance of Mediator. As service consumers send requests, Mediator distributes messages to several different service provider endpoints that are configured in the VSD. For information about creating a load-balanced virtual service, see the sections

Virtualized Services in CentraSite Control or *Virtualizing APIs Using the CentraSite Business UI* of the CentraSite documentation.

Note: A virtual service that uses a load balancing protocol does not require a load balancer like those used in a cluster. For more information about load balancers in a cluster, see "Load Balancers" on page 31.

This type of load balancing scenario looks as follows:



Round-Robin Distribution

When the virtual service is configured using a load balanced routing type, a single Mediator takes requests and routes them to the service providers in turn, using a round-robin distribution. As Mediator receives requests, it distributes the request to the service provider whose turn it is to process the message.

Note: Mediator does not currently support load balancing requests based on which service provider is the highest performing.

Inactive Endpoints

If a service provider endpoint is down when Mediator tries to distribute a request to it, Mediator considers this endpoint inactive. The endpoint will remain considered inactive for 60 seconds, during which time all subsequent invocations skip that endpoint and are instead routed to the remaining endpoints.

If all of the endpoints are down, then Mediator considers all of the endpoints inactive for 60 seconds and sends an error event to CentraSite (if enabled).

Note: Any subsequent invocations during the 60-second inactive period do not generate an error event. This means that Mediator will send only one error event for each inactive period. However, Mediator will return service faults for all of the failed requests.

4 The Built-In Run-Time Actions

Summary of the Built-In Run-Time Actions

This section provides a summary of the run-time actions. For complete details of all actions, see the *Run-Time Governance Reference* section in the CentraSite documentation.

There are two separate sets of run-time actions you can use:

- Run-time actions for virtual services.

  You use these actions only when you are using CentraSite Control to create run-time policies for virtual services. See "The Built-In Run-Time Actions for Virtual Services" on page 44.

- 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. See "The Built-In Run-Time Actions for Virtualized APIs" on page 47.

The Built-In Run-Time Actions for Virtual Services

This section provides a summary of the run-time actions 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.

A run-time policy provides run-time governance capabilities to a virtual service. A *run-time policy* is a sequence of actions that are carried out by Mediator when a consumer requests a particular service through Mediator.

The actions in a policy perform activities such as identifying/authenticating consumers, validating digital signatures and capturing performance measurements. An *action* is a single task that is included in a run-time policy and is evaluated by Mediator at run time. Actions have one or more parameters, which you configure when you insert the actions into a policy. For example, an action that identifies consumers specifies one or more identifiers to identify the consumers who are trying to access the services.

Mediator provides built-in action templates. A built-in action template is a definition of an action that can be used in a policy. An action template specifies the set of parameters associated with a particular policy action. You can use these action templates to create actions for Mediator.

When you create run-time policies in CentraSite, you:

1. Specify the service(s) to which the policy should apply.
2. Add the desired actions to the policy, and configure their parameters.
3. Activate the policy when you are ready to put it into effect. When you deploy the virtual services to which the policy is applied, the policy will also be deployed.

Following is a summary of the built-in actions you can include in a run-time policy. For complete details of all actions, as well as common usage cases for identifying/

authenticating consumers, see the *Run-Time Governance* section in the CentraSite documentation.

- "WS-SecurityPolicy 1.2 Actions" on page 45
- "Monitoring Actions" on page 45
- "Additional Actions" on page 46

For the procedure to create a run-time policy, see the *Virtual Services in CentraSite Control* section of the CentraSite documentation.

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 authentication actions to verify that the requests for virtual services contain a specified WS-SecurityPolicy element. Mediator provides the following actions:

- **Require WSS SAML Token**: Requires that a WSS Security Assertion Markup Language (SAML) assertion token be present in the message header to validate to validate service consumers.
- **Require WSS Username Token**: Requires that a WSS username token and password be present in the message header to validate service consumers.
- **Require WSS X.509 Token**: Requires that a WSS X.509 token be present in the message header 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. Mediator provides the following actions:

- **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 *Service Level Agreement* (SLA), which is set of conditions that defines the level of performance that a specified consumer should expect from a service.
- **Throttling Traffic Optimization**: 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, etc.

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, this action can be used 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.

  Note: Mediator also supports NTLM and OAuth2 authentication, which you can specify in a virtual service's Routing Protocol step. For details, see the *Virtualized Services in CentraSite Control* section of the CentraSite documentation.

- **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 Application assets that are registered in CentraSite as consumers for the service.
- **Log Invocations**: This action logs request/response payloads to a destination you specify.
- **Validate Schema**: This action validates all XML request and/or response messages against an XML schema referenced in the WSDL.

For complete details of all actions, as well as common usage cases for identifying/authenticating consumers, see the *Run-Time Governance Reference* section in the CentraSite documentation.

The Built-In Run-Time Actions for Virtualized APIs

This section provides a summary of the run-time actions you can include in the policy enforcement rules for a virtualized API. You use these actions only when you are using the CentraSite Business UI to create policy enforcement rules for virtualized APIs

A policy enforcement rule provides run-time governance capabilities to a virtualized API. A *run-time policy* is a sequence of actions that are carried out by Mediator when a consumer requests a particular API through Mediator.

The actions in a policy perform activities such as identifying/authenticating consumers, validating digital signatures and capturing performance measurements. An *action* is a single task that is included in a policy enforcement rule and is evaluated by Mediator at run time. Actions have one or more parameters, which you configure when you insert the actions into a rule. For example, an action that identifies consumers specifies one or more identifiers to identify the consumers who are trying to access the APIs.

Following is a summary of the built-in run-time actions you can include in a policy enforcement rule. For complete details of all actions, see the *Run-Time Governance* section of the CentraSite documentation.

- "Request Handling Actions" on page 47
- "Policy Enforcement Actions" on page 48
- "Response Handling Actions" on page 51
- "Error Handling Action" on page 51

Request Handling Actions

Mediator provides the following actions for handling requests:

- **Require HTTP/HTTPS**: Specifies the protocol (HTTP or HTTPS), SOAP format (for a SOAP-based API), and the HTTP methods (for a REST-based API) for the virtual API to accept requests.
- **Require JMS**: Specifies the JMS protocol to be used for the API to accept and process the requests.
- **Request Transformation**: Invokes an XSLT transformation file in the request before it is submitted to the native API.
- **Invoke webMethods Integration Server Service**: Invokes a webMethods IS service to pre-process the request before it is submitted to the native API.

Policy Enforcement Actions

Mediator provides the following categories of policy enforcement actions:

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 Invocations**: 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 (e.g., 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**: Routes requests to different endpoints based on specific values that appear in the request message.
- **Content Based Routing**: Routes 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 etc.), 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.

There are two different lists of consumers in Mediator:

- **List of Registered Consumers**: Registered consumers are those users and consumer applications (represented as Application assets) who are available in Mediator and who are also registered as consumers for the API in CentraSite.
- **List of Global Consumers**: Global consumers are those users and consumer applications (represented as Application assets) who are available in 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:

- Identify the clients who are trying to access the APIs (through IP address or hostname).
- Validate the client's credentials.

Following is the list of security actions:

- **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 WSS Username Token**: Applicable only for SOAP APIs.
  - 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**: Applicable only for SOAP APIs.
  - 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**:
  - 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**: Applicable only for SOAP APIs. Requires that a request's XML element (which is represented by an XPath expression) be encrypted.
- **Require Signing**: Applicable only for SOAP APIs. Requires that a request's XML element (which is represented by an XPath expression) be signed.
- **Require SSL**: Applicable only for SOAP APIs. Requires that requests be sent via SSL client certificates.
- **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, etc.

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 file in the response payloads from XML format to the format required by the client.
- **Invoke webMethods Integration Server Service**: Invokes a webMethods Integration Server service to process the response from the native API before it is returned to the client.

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.

- **Custom SOAP Response Message**: 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.

For complete details of all actions, see the *Run-Time Governance Reference* section in the CentraSite documentation.

5 Configuring Mediator

Overview

Mediator enforces the policies you apply to virtual services in CentraSite. For Mediator to enforce these policies, you define parameters for:

Configuration Task	Description
CentraSite communication configuration	You must define the communication parameters required for Mediator to exchange data with CentraSite. See "Configuring Communication with CentraSite " on page 56.
EDA destinations for publishing run-time events and metrics	Mediator can use EDA to publish run-time events and metrics to a database or a messaging server such as webMethods Universal Messaging. See "EDA Configuration for Publishing Run-Time Events and Metrics" on page 57.
SNMP server destinations for publishing run-time events and metrics	Alternatively, Mediator can use the CentraSite SNMP server or a third-party SNMP server to publish run-time events and metrics. See "SNMP Destinations for Run-Time Events" on page 60.
SMTP server destinations for sending alerts and logging transaction payloads	You can configure Mediator to: ■ Send monitoring alerts to an SMTP email server when user-specified performance conditions are violated. ■ Log the payloads of all transactions to an SMTP email server. See "SMTP Destinations for Alerts and Transaction Logging" on page 67.
Load balancing URL configuration	Load balancing enables Mediator to distribute messages it receives between a set of listed endpoints. You can set Mediator to use either HTTP or HTTPS protocols for load balancing. See "Load Balancing Configuration" on page 69.
Keystore and truststore configuration	Keystores and truststores are required for message-level security. They provide SSL authentication, encryption/decryption, and digital signing/verification

Configuration Task	Description
	services for all message content that Mediator sends. See "Keystore Configuration" on page 70.
Ports configuration	You can specify one or more HTTP or HTTPS ports on which Mediator and the deployed virtual service will be available. See "Ports Configuration" on page 71.
Global service fault response configuration	Configure the format and content of global service fault responses that are returned to consuming applications. See "Configuring Global Service Fault Responses" on page 72.
SAML support in Mediator	You can configure Mediator to act as a Security Token Service (STS) client. See "SAML Support in Mediator " on page 77.
WS-Addressing configuration	Implement WS-Addressing using Mediator, so that clients can send WS-Addressing information to native services. See "WS-Addressing Processing in Mediator " on page 89.
GZIP configuration	Reduce the volume of data that is sent by native services' SOAP responses. Mediator can compress the responses based on the transport encoding (the Accept-Encoding and Content-Encoding headers). See " Mediator's GZIP Functionality" on page 98.
Custom Content-Type configuration	You can specify custom Content-Types for REST services. See "Configuring Custom Content-Types" on page 102.
OAuth2 inbound processing	Describes how to configure your system for OAuth2 inbound processing. See "OAuth2 Inbound Configuration" on page 103.

In addition, you can view the services and consumer applications that are deployed to Mediator. See "Viewing the Services Deployed to Mediator " on page 76 and "Viewing the Consumer Applications Deployed to Mediator " on page 76.

Before Configuring Mediator

This section describes actions you should perform before configuring Mediator.

1. Install webMethods Mediator as part of the webMethods Integration Server installation. For information about installing Mediator, see *Installing webMethods and Intelligent Business Operations Products*.
2. Ensure that you have webMethods administrator privileges so that you can access Mediator's administrative screens. For information about setting user privileges, see *webMethods Integration Server Administrator's Guide*.
3. Make sure you have defined one or more Mediator targets in CentraSite.

   A *target* is an object that represents a specific instance of the Mediator target type. You can have multiple instances of the Mediator target type. For example, if your SOA environment includes three instances of Mediator, your CentraSite registry/repository will include three targets; one for each Mediator instance. For information about defining targets in CentraSite, see the section *Run-Time Targets* in the CentraSite documentation.
4. For each Mediator target that you need to configure, add your user name to the user group in CentraSite called "<*TargetName*\> Synchronization Group".

   CentraSite automatically creates this group when a target is created in CentraSite. As a member of this group, you will automatically receive instance-level permissions for all virtual services that are deployed on the target. However, note the following:

   Important: Your user name in this group must have either one of the following CentraSite roles and permissions: The "Asset Provider" role OR the "Manage Assets" permission.
5. Start Integration Server and Integration Server Administrator, if they are not already running. For information about starting Integration Server and Integration Server Administrator, see *webMethods Integration Server Administrator's Guide*.

Configuring Communication with CentraSite

To enable Mediator to exchange data with CentraSite

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > CentraSite Communication**.
3. On the **CentraSite Configuration** page, click **Edit**.
4. Set the **CentraSite UDDI Publisher Configuration** parameters as follows and click **Save**:

For this parameter...	Specify...
Protocol	HTTP (the default) or HTTPS.

For this parameter...	Specify...
Host Name	The host name or IP address of the machine on which CentraSite is running.
Target Name	The name of the Mediator instance configured as a target in CentraSite. This name must match the target name defined in CentraSite (in CentraSite Control, go to **Operations > Targets** to view the target list).
UDDI Port	The port used for UDDI access to CentraSite. The default is `53307`.
User Name/ Password	The CentraSite user name and password that Mediator must use to access CentraSite. If you are using the Operating System auth mechanism, use the following format for the user name: *CS-Host-Name \CS-user-name*
Report Performance Data	Specify whether Mediator should collect and report performance metrics to CentraSite. The default is true. **Note:** In order to save the configuration parameters, you must select the **Report Performance Data** check box before you click **Save**. You can then uncheck it and click **Save** if you do not want to report the performance data.
Publish Interval (minutes)	Specify how often (in minutes) Mediator should report performance metrics. Enter a value from 1 through 60. The default is 60. For more information, see "The Metrics Tracking Interval" on page 26.

EDA Configuration for Publishing Run-Time Events and Metrics

Mediator can use EDA to publish data about run-time events and key performance indicator (KPI) metrics to a database or a messaging server such as webMethods Universal Messaging. Mediator uses the EDA capability of Universal Messaging to publish the event type(s) that you select, to the specified destinations. You can also use the Guaranteed Delivery of Events provided by EDA to ensure that the events generated are not lost. For more details on EDA and configuring guaranteed delivery, see *Implementing Event-Driven Architecture with Software AG Products*.

Alternatively, instead of using EDA to publish the events and metrics, you can use an SNMP server (see "SNMP Destinations for Run-Time Events" on page 60).

On the EDA Configuration page you need to specify:

- The types of run-time events that Mediator should publish.
- Whether the key performance indicator (KPI) metrics should be published and the intervals at which to publish them.
- The destination to which Mediator should publish the events and KPI metrics.

Prerequisites:

- If you are publishing the Mediator events to the database, ensure that a JDBC connection pool is defined in Integration Server which can be used by Mediator. JDBC connection pools are specified in the Integration Server Administrator's **Settings > JDBC Pools** page.

  Note: If you change the definition of the JDBC pool configured for EDA and point it to another database, ensure that you select the corresponding connection pool alias in the **EDA Configuration** page and save the configuration.

- If you are publishing the Mediator events to a messaging server such as webMethods Universal Messaging, you must install and start the messaging server before you start Integration Server.

If you select both the destinations: database, webMethods Universal Messaging, Mediator publishes the events and KPI metrics first to webMethods Universal Messaging and thereafter to the database. However, if Mediator is not able to publish the events and KPI metrics to webMethods Universal Messaging, it will not publish them to the database as well.

If a JDBC connection pool is not defined in Integration Server, you must define one as described in the section *Connect Products to Database Components* in the document *Installing webMethods and Intelligent Business Operations Products*. Be sure to perform both of the following procedures within that section:

- *Define an Integration Server Connection Pool*.
- *Point an Integration Server Function at an Integration Server Function Pool*.

Note: You cannot use Integration Server's embedded database for publishing events and metrics.

To configure Mediator to publish events and metrics to an EDA destination

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > EDA Configuration**.
3. On the EDA Configuration screen, click **Edit**.
4. Under **Event Types**, complete the fields as follows:

Event Type	Description
Lifecycle	Specify whether to publish Lifecycle events. A Lifecyle event occurs each time Mediator starts or shuts down.
Error	Specify whether to publish Error events. An Error event occurs each time an invocation of a virtual service results in an error.
Policy Violation	Specify whether to publish Policy Violation events. 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.

5. Under **Performance Metrics**, select the **Publish Interval** checkbox and enter a value from 1 through 60 in the field next to it. The default is 60. This setting specifies how often (in minutes) Mediator should report performance metrics. For more information, see "The Metrics Tracking Interval" on page 26.
6. Under **Destinations**, specify a destination to publish the events and KPI metrics.

Event Type	Description
Emit to Database	Specify whether Mediator should publish the events and KPI metrics to the database (that is, a JDBC connection pool that has been defined for Mediator in Integration Server).
Connection Pool alias	If you have selected the **Emit to Database** option, specify the JDBC connection pool which has been defined for Mediator in Integration Server. **Note:** JDBC connection pools are specified in Integration Server Administrator's **Settings > JDBC Pools** page.
Emit to Default EDA Endpoint	Specify whether Mediator should publish the events and KPI metrics to the default EDA endpoint. If you select this option Mediator publishes the events to the Universal Messaging event bus.

For details on enabling EDA for events in CentraSite, see *CentraSite Administrator's Guide*.

Note: You can set the limit for the number of entries (events) that NERV will keep in the cache by setting the parameter `pg.nerv.PgMenConfiguration.cache.size` in

the pg-config.properties file. By default, the limit is `2000`. For details, see "Advanced Settings" on page 121.

7. Click **Save**.

   Your changes take effect immediately.

SNMP Destinations for Run-Time Events

Mediator can use SNMP traps to report run-time events and metrics to an SNMP server. You can use either (or both) of the following kinds of SNMP servers:

- **The CentraSite SNMP server**, which uses the SNMPv3 user-based security model. To set this server as a destination, see "Setting Mediator to Use the CentraSite SNMP Server" on page 60.
- **A third-party SNMP server.** You can set Mediator to use either the SNMPv1 community-based security model (the default), or the SNMPv3 user-based security model. To set this server as a destination, see "Setting Mediator to Use a Third-Party SNMP Server" on page 63.

In addition, you must specify the events types you want to publish as described in "Specifying the Events to Publish" on page 66 and configure the **Target Name** in the CentraSite Communication page as described in "Configuring Communication with CentraSite " on page 56. Mediator sends the events to the SNMP server along with information on the target(s). If no target is configured for an event, the event will be persisted in the CentraSite database with the target key value `unknown`.

Note: Alternatively, instead of using an SNMP server to publish the events and metrics, you can use EDA (see "EDA Configuration for Publishing Run-Time Events and Metrics" on page 57).

Setting Mediator to Use the CentraSite SNMP Server

You can set Mediator to use the CentraSite SNMP server, which uses SNMPv3 user-based security model. The CentraSite server must have an SNMP receiver running in order to receive SNMP notifications from Mediator.

Mediator's server-side SNMP configuration must be in sync with CentraSite's client-side configuration. By default, the configuration defaults are synched up. CentraSite's configuration is statically defined in a war file in CentraSite's event receiver, located here:

CentraSite_directory /cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml

For details, see the section *Run-Time Governance Reference > Run-Time Events and Key Performance Indicator (KPI) Metrics > Configuring CentraSite to Receive Run-Time Events and Metrics* in the CentraSite documentation.

To set Mediator to use the CentraSite SNMP server

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > SNMP**.
3. On the **SNMP Configuration** screen, click **Edit**.
4. Under **CentraSite SNMP Configuration**, select the **Send TRAPs to CentraSite** check box.
5. Set the **CentraSite SNMP Configuration** parameters as follows:

For this parameter...	Specify...
Host Name/IP Address	The host on which the CentraSite server is installed and running. The value must match the value of the property `com.softwareag.centrasite.soalink.events.snmp.host` in CentraSite's Event Listener configuration file <*CentraSite_directory*>/cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml. The server name cannot be `localhost` or the loopback (127.0.0.1). For details, see the section *Run-Time Governance Reference > Run-Time Events and Key Performance Indicator (KPI) Metrics > Configuring CentraSite to Receive Run-Time Events and Metrics* in the CentraSite documentation.
Port	The port to which the CentraSite SNMP listener is bound. The value must match the value of the property `com.softwareag.centrasite.soalink.events.snmp.port` in CentraSite's Event Listener configuration file <*CentraSite_directory*>/cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml. The default is 8181. **Note:** 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.
Transport	The protocol used by SNMP to send traps. The value must match the value of the property `com.softwareag.centrasite.soalink.events.snmp.`

For this parameter...	Specify...
	`transport` in CentraSite's Event Listener configuration file <*CentraSite_directory*>/cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml. Valid values are `TCP` (default) and `UDP`.
User Name	The SNMPv3 user name to use when connecting to the receiver. The value must match the value of the property `com.softwareag.centrasite.soalink.events.snmp.securityName` in CentraSite's Event Listener configuration file <*CentraSite_directory*>/cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml.
Use Authorization (optional)	Whether an authorization key is required. It is set to off by default. You cannot edit the authorization fields unless **Use Authorization** is selected.
Authorization Password / Retype Authorization Password	The key to be used for authorization. The value must match the value of the property `com.softwareag.centrasite.soalink.events.snmp.authPassPhraseKey` in CentraSite's Event Listener configuration file <*CentraSite_directory*>/cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml.
Authorization Protocol	The authorization protocol to use. The value must match the value of the property `com.softwareag.centrasite.soalink.events.snmp.authProtocol` in CentraSite's Event Listener configuration file <*CentraSite_directory*>/cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml. Valid values are `MD5` (default) or `SHA`.
Use Privacy (optional)	Whether a privacy (encryption) key is required. It is set to off by default. You cannot edit the privacy fields unless **Use Privacy** is selected.
Privacy Password / Retype Privacy Password	The key to be used for privacy. The value must match the value of the property `com.softwareag.centrasite.soalink.events.snmp.privPassPhraseKey` in CentraSite's Event Listener configuration file <*CentraSite_directory*>/cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml.

For this parameter...	Specify...
Privacy Protocol	The privacy protocol to use. The value must match the value of the property `com.softwareag.centrasite.soalink.events.snmp.privProtocol` in CentraSite's Event Listener configuration file <*CentraSite_directory*>/cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml. Valid values are `DES` (default), `AES128`, `AES192`, `AES256`, `3DES`, and `DESEDE`.

6. Click **Save**.

   Your changes take effect immediately.

Note: Additional parameters are available in the Mediator properties file; see the section *pg.cs.snmpTarget* in "Advanced Settings" on page 121.

Setting Mediator to Use a Third-Party SNMP Server

In order to use a third-party SNMP server, you must:

- Import into your third-party SNMP server the predefined MIB definition for Mediator's traps. This MIB describes the format of the traps (see "Importing the MIB for Mediator's Traps" on page 63).
- Configure Mediator to use a third-party SNMP server. Mediator supports:
  - The SNMPv3 user-based security model (see "Setting Mediator to Use a Third-Party SNMP Server (SNMP v1 Community-Based Security Model)" on page 65).
  - The SNMPv1 community-based security model (see "Setting Mediator to Use a Third-Party SNMP Server (SNMP v3 User-Based Security Model)" on page 64).

Importing the MIB for Mediator's Traps

To import the MIB for Mediator's traps

Import webMethodsESB.MIB into your third-party SNMP server from the following directory:

Integration Server_directory \instances*instance_name* \packages\WmMediator\config \resources

Note: For information about importing the MIB into your specific SNMP server, see the documentation for that product. For more information about the webMethodsESB.MIB

file, see the section *Run-Time Targets > Creating and Managing Target Types* of the CentraSite documentation.

Setting Mediator to Use a Third-Party SNMP Server (SNMP v3 User-Based Security Model)

To set Mediator to use the SNMP v3 user-based security model

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > SNMP**.
3. On the **SNMP Configuration** screen, click **Edit**.
4. Under **3rd Party SNMP Configuration**, select the **Send TRAPs to 3rd party SNMP Server** check box.
5. For **SNMP Target Type**, click **User**.
6. Set the **3rd Party SNMP Configuration** parameters as follows:

For this parameter...	Specify...
Host Name/IP Address	The SNMP server's exact IP address that is specified in CentraSite, or a hostname that resolves to that IP address. The host name cannot be `localhost` or the loopback (127.0.0.1).
Port	The SNMP trap receiver port that is listening for SNMP requests and packets.
Transport	The protocol used by SNMP to send traps. Valid values are `TCP` (default) and `UDP`.
User Name	The SNMPv3 user name to use when connecting to the receiver.
Use Authorization (optional)	Specifies whether an authorization key is required. It is set to off by default. You cannot edit the authorization fields unless **Use Authorization** is selected.
Authorization Password / Retype Authorization Password	The key to be used for authorization. This must also be configured on the SNMP server.

For this parameter...	Specify...
Authorization Protocol	The authorization protocol to use. Valid values are `MD5` (default) or `SHA`
Use Privacy (optional)	Whether a privacy (encryption) key is required. It is set to off by default. You cannot edit the privacy fields unless **Use Privacy** is selected.
Privacy Password	The key to be used for privacy.
Retype Privacy Password	The same password you entered in the **Privacy Password** field to verify that you entered it correctly.
Privacy Protocol	The privacy protocol to use. Valid values are `DES` (default), `AES128`, `AES192`, and `AES256`.

7. Click **Save**.

   Your changes take effect immediately.

Note: Additional parameters are available in the Mediator properties file; see the sections *pg.snmp.userTarget* and *pg.snmp.customTarget* in "Advanced Settings" on page 121.

Setting Mediator to Use a Third-Party SNMP Server (SNMP v1 Community-Based Security Model)

To set Mediator to use the SNMP v1 community-based security model

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > SNMP**.
3. On the **SNMP Configuration** screen, click **Edit**.
4. Under **3rd Party SNMP Configuration**, select the **Send TRAPs to 3rd party SNMP Server** check box.
5. For **SNMP Target Type**, click **Community**.
6. Provide the following information:

For this parameter...	Specify...
Host Name/IP Address	The name or IP address of the third-party SNMP server. The host name cannot be `localhost`.
Port	The SNMP trap receiver port that is listening for SNMP requests and packets. The default is `2162`.
Transport	The protocol used by SNMP to send traps. Valid values are `TCP` (default) and `UDP`. If you select `UDP`, ensure that the SNMP server is in the same subnet, or else configure the routers to get packets across subnet boundaries. The maximum PDU size when running in UDP mode is 64Kb, which might be restrictive when sending large request and response payloads for Transaction Events.
Community Name	The community name to be used to interact with the SNMP receiver. This value must match the value that you set in the SNMP receiver. The default is `public`.

7. Click **Save**.

   Your changes take effect immediately.

Note: Additional parameters are available in the Mediator properties file; see the sections *pg.snmp.communityTarget* and *pg.snmp.customTarget* in "Advanced Settings" on page 121.

Specifying the Events to Publish

In addition to configuring an SNMP destination, you must specify which events types to publish: Lifecycle, Error and/or Policy Violation events. For more information about event types, see "Key Performance Indicator Metrics and Run-Time Event Notifications" on page 21.

To specify the events to publish

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > SNMP**.
3. On the **SNMP Configuration** screen, click **Edit**.
4. Under **Event Generation**, select one or more of the event types that you want to publish:

Event Type	Description
Lifecycle	A Lifecyle event occurs each time Mediator starts or shuts down.
Error	An Error event occurs each time an invocation of a virtual service results in an error.
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.

5. Click **Save**.

   Your changes take effect immediately.

SMTP Destinations for Alerts and Transaction Logging

You can configure Mediator to:

- Send monitoring alerts to an SMTP email server when user-specified performance conditions are violated. To do this, you can include the following actions in your virtual service's run-time policy:
  - "Monitor Service Performance".
  - "Monitor Service Level Agreement".
  - "Throttling Traffic Optimization".
- Log the payloads of all transactions to an SMTP email server, using the "Log Invocations" action. Mediator sends the payloads as email attachments that are compressed using gzip data compression.

To enable Mediator to publish send alerts and to log transaction payloads to an SMTP server, use the following procedure.

To configure an SMTP destination

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > Email**.
3. On the **Email Configuration** screen, click **Edit**.
4. Under **Email Configuration**, set the parameters as follows:

For this parameter...	Specify...
SMTP Host Name/ IP Address	The address of the SMTP server to be used by Mediator for sending email alerts.
Port	The port on which the SMTP server is listening.
User	The user name of the email account used to log into your SMTP server.
Password	The password of the email account used to log into your SMTP server.
Retype Password	The same password you entered in the **Password** field.
From	The email address that will appear in the From field of the event email. If you leave this field blank, Mediator generates a default email address composed of the configured target name and hostname of the Integration Server, as follows: `Target-Name@hostname`
TLS Enabled	Whether to use transport-layer security (TLS). If selected, the truststore configured for Mediator must include a certificate in the email server's certificate chain. This requires that both a keystore and truststore be configured for Integration Server. For information about configuring keystores and truststores for Integration Server, see the *Securing Communications with the Server* section in the document *webMethods Integration Server Administrator's Guide*. **Note:** You must also specify a value for the **IS Truststore Name** field in the Mediator Configuration screen (see "Keystore Configuration" on page 70).
Test Recipient (optional)	The email address of the person who should receive the test email.

5. To test your connection, enter an email address in the **Test Recipient** field and click **Test**.

If the configuration is...	Then...
Successful	Mediator sends an email to the address you entered in the **Test Recipient** field and displays a verification message.
Not successful	Mediator displays an error message. Reconfigure your email settings and try again.

6. Click **Save**.

   Your changes take effect immediately.

   Note: Additional parameters are available in the Mediator properties file; see the section *pg.email* in "Advanced Settings" on page 121.

Load Balancing Configuration

If you cluster Mediator instances, you must configure Mediator with the load balancer URL appropriately for Mediator to report the virtual service address at the load balancer URL instead of having direct access address withMediator. You can use either HTTP or HTTPS protocols for load balancing.

To configure load balancer URLs

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > General**.
3. On the **Mediator Configuration** screen, click **Edit**.
4. Under **HTTP Config**, set the parameters as follows:

For this parameter...	Specify...

Load Balancer URL (HTTP)	The primary HTTP load balancer URL and port number to use.
	For the URL, you can specify either the IP address or host name of the load balancer with the port number, as follows:
	`http://IP-address:portnumber`
	or
	`http://hostname:portnumber`

For this parameter...	Specify...
	If specified, all the virtual services hosted in Mediator will use this value.

Load Balancer URL (HTTPS)	The primary HTTPS load balancer URL and port number to use.
	
	For the URL, you can specify either the IP address or host name of the load balancer with the port number, as follows:
	
	`https://IP-address:portnumber`
	
	or
	
	`https://hostname:portnumber`
	
	If specified, all the virtual services hosted in Mediator (and exposed on HTTPS) will use this value.
	
	Note: The SSL connection will be dropped if the remote server hosting the service is not trusted by Integration Server. To avoid this, you must create an Integration Server truststore and select it as part of the keystore configuration. For information about configuring a truststore for Integration Server, see *webMethods Integration Server Administrator's Guide*. For information about selecting the truststore as part of the Mediator configuration, see "Keystore Configuration" on page 70.

5. Click **Save**.

   Your changes take effect immediately.

Keystore Configuration

Keystores and truststores are required for message-level security. They provide SSL authentication, encryption/decryption, and digital signing/verification services for all message content that Mediator sends. You must first set up keystores and truststores in Integration Server (see the *Securing Communications with the Server* section in the document *webMethods Integration Server Administrator's Guide*) and then configure the keystore information here.

To configure keystore information

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > General**.

3. On the **Mediator Configuration** screen, click **Edit**.
4. Under **Keystore Config**, set the parameters as follows:

For this parameter...	Specify...
IS Keystore Name	The Integration Server keystore that Mediator should use. This field lists all available Integration Server keystores. If you have not configured an Integration Server keystore, the list will be empty.
Alias (signing)	The signing alias. This alias is the value that is used to sign the outgoing response from Mediator to the original service consumer. It is auto-populated based on the keystore selected from the **IS Keystore Name** above. This field will list all the aliases available in the chosen keystore. If there are no configured keystores, this field will be empty.
IS Truststore Name	The Integration Server truststore used for establishing the HTTPS connection to the target service provider. It should contain the certificate of the service provider. **Note:** If you selected **TLS Enabled** in "SMTP Destinations for Alerts and Transaction Logging" on page 67, you must select the truststore that contains either the email certificate or the certificate authority of the email server.

5. Click **Save**.

   Your changes take effect immediately.

Ports Configuration

You can specify one or more HTTP or HTTPS ports on which Mediator and the deployed virtual services will be available.

Mediator is always available on the primary HTTP port. The primary HTTP port is the port specified on the Integration Server's **Security > Ports** page. You can specify additional HTTP and HTTPS ports on Mediator's Ports Configuration page. For information about configuring ports for Integration Server, see the *Configuring Ports* section in the document *webMethods Integration Server Administrator's Guide*.

Important: If you specify multiple ports, the port that is reported in the WSDL is the *non-primary port with the lowest number*. There is only one port reported in the WSDL retrieved through Mediator, but all the specified ports are usable.

If you are using a webMethods Enterprise Gateway (EG) Registration port to process requests from external clients, specify the EG port in Mediator so that the EG server can access it. If CentraSite deploys virtual services to Mediator through Enterprise Gateway, then the target deployment endpoint should be configured with the EG host and port.

To specify the ports on which Mediator is available

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > General**.
3. On the **Mediator Configuration** screen, click **Edit**.
4. Under **Ports Config** set the parameters as follows:

For this parameter...	Specify...
HTTP Ports Configuration	The HTTP port(s) on which Mediator and the deployed virtual services will be available.
HTTPS Ports Configuration	The HTTPS port(s) on which Mediator and the deployed virtual services will be available.

5. Click one or more ports in the Available Ports list and click **Add**.

   If any of the previously selected ports have been disabled or removed, they will not appear in the Available Ports list or the Selected Ports list.

6. Click **Save**.

   Your changes take effect immediately.

Configuring Global Service Fault Responses

You can use these options to configure global error responses for all virtual services or virtual APIs.

Alternatively, you can configure each virtual service or virtual API individually, as follows:

- For a virtual service, you can configure an Error Messaging step on the virtual service's "Response Processing" tab, as described in the section *Virtualized Services in CentraSite Control > Configuring Virtual Services* of the CentraSite documentation.

- For a virtual API, you can configure a "Custom SOAP Response Message" action for the virtual API, as described in the section *Virtualizing APIs Using the CentraSite Business UI* > *Configuring Virtual Services* of the CentraSite documentation.

The precedence of the error or response messaging instructions is as follows:

- If you configure an Error Messaging step or a "Custom SOAP Response Message" action, that step or action takes precedence over any settings on the global Service Fault Configuration page.
- If you do not configure an Error Messaging step or a "Custom SOAP Response Message" action, the settings on the global Service Fault Configuration page take precedence.

Note: By default, when Mediator receives 4xx error codes from the native services, Mediator sends 500 error codes to the client. If you want Mediator to send the 4xx error codes to the client, go to:

Integration Server_directory \instances*instance_name* \packages\WmMediator\config \resources\beans\pg-core.xml

and set the `handleClientErrorCode` property for the following bean to true:

```
<bean id="httpResponseCodeCallback"
class="com.softwareag.pg.axis2.transports.ISHTTPResponseCodeCallback">

<property name="handleClientErrorCode" value="true"/></bean>
```

When the `handleClientErrorCode` property is set to true, the following native service response error codes will be sent to the client: 400, 401 (if the request returns the user), 403, 406, 407, 408, 409, 412, 413, 415, 416, 417. For all other error codes, error code 500 will be sent back to the client.

To configure global service fault responses for all virtual services

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Administration > Service Fault**.
3. On the **Service Fault Configuration** screen, click **Edit** and complete the following fields.

Field	Description
Default Fault 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".`

Field	Description
	The fault response contains predefined fault handler variables, which are described in "The Fault Handler Variables" on page 75. This fault response is returned in both of the following cases: ■ In cases where a fault is returned by the native service provider. In this case, the `$ERROR_MESSAGE` variable in the fault response will contain the message produced by the provider's exception that caused the error. Mediator discards the native service provider's fault and does not return this content to the web service caller since it could be considered a security issue, especially if the native provider is returning a stack trace with its response. ■ In cases where a fault is returned by internal Mediator exceptions (policy violation errors, timeouts, etc.). In this case, the `$ERROR_MESSAGE` variable will contain the error message generated by Mediator. 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: ■ Mediator's predefined context variables. ■ Custom context variables that you declare using the Mediator API. For details about the predefined and custom context variables, see the section *Virtualized Services in CentraSite Control > Using Context Variables in Virtualized Services* in the CentraSite documentation.
Send Native Provider Fault	When you select this option, Mediator sends the native service provider's fault content, if available. Mediator will send whatever content it received from the native service provider. If you select this option, the **Default Fault Response** is ignored when a fault is returned by the native service provider. (Faults returned by internal Mediator exceptions will still be handled by the **Default Fault Response** option.)

4. Click **Save**.

Note: Unlike with SOAP specifications, there is no agreed upon format to suggest an error condition for XML and REST services (that is, there is no element nested in a `</soap:Fault>` element nested in a `</soap:Body>`). Mediator assumes that XML and REST services will follow HTTP conventions and return responses with return codes in the 200-299 range when service calls are successful. This is the only way Mediator can determine that a native provider's response should be interpreted as a failure.

The Fault Handler Variables

The following fault handler variables are used in fault response messages. You can use these variables in the global Service Fault Configuration page and in a virtual service's "Response Processing" step.

In addition, you can use Mediator's context variables, which are described in the section *Virtualized Services in CentraSite Control > Using Context Variables in Virtualized Services* in the CentraSite documentation.

Error Handler Variable	Description
`$ERROR_MESSAGE`	When a fault is returned by the native service provider, this variable will contain the message produced by the provider's exception that caused the error. This is equivalent to the `getMessage` call on the Java Exception. This maps to the faultString element for SOAP 1.1 or the Reason element for SOAP 1.2 catch. For REST service calls, the message is returned inside an `</Exception>` tag. If the response is XML, the message is returned inside `<Exception>'custom message'</Exception>`. If the response is JSON, it will be returned inside `{"Exception":"Invalid response"}`. When a fault is returned by internal Mediator exceptions (policy violation errors, timeouts, etc.), this variable will contain the error messages generated by Mediator.
`$OPERATION`	The operation that was invoked when this error occurred.
`$SERVICE`	The service that was invoked when this error occurred.

Error 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.
`$CONSUMER_APPLICATION`	The currently identified consumer application. If the service's policy does not contain the "Identify Consumer" action, `$CONSUMER_APPLICATION` will return "null".

Viewing the Services Deployed to Mediator

The **List of Mediator Services** screen shows all services currently deployed to Mediator. You can view the virtual service definition (VSD) and WSDL file of each service.

To view the services deployed to Mediator

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Services**.
3. For any service in the list, click **VSD** to view the virtual service definition or **WSDL** to view the WSDL file for the service.

Viewing the Consumer Applications Deployed to Mediator

The **List of Mediator Consumers** screen shows all consumer applications configured in CentraSite for your system. You can use this screen to view the details of the consumer applications configured in CentraSite.

To view the consumer applications deployed to Mediator

1. Open the Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Solutions > Mediator > Consumers**.

3. View the details of consumer applications synchronized with CentraSite. You can view the details of all applications or selected applications.

SAML Support in Mediator

Mediator supports the following SAML subject confirmation methods:

- SAML 1.1 and 2.0 Holder-of-Key tokens (see "Configuring for SAML Holder-of-Key Processing" on page 77).
- SAML 2.0 Sender-Vouches tokens (see "Configuring for SAML Sender-Vouches Processing" on page 82).
- SAML 1.1 Bearer tokens (see "Configuring for SAML Bearer Token Processing" on page 86).

Regardless of which SAML subject confirmation method you use, you must first configure an Integration Server keystore, as described below.

Configuring Integration Server Keystores

You must configure an Integration Server keystore to enable Mediator to process any supported SAML token (Holder-of-Key tokens, Sender-Vouches tokens or Bearer tokens).

To configure an Integration Server keystore

1. In Integration Server, create a new keystore alias, as described in the *Keystores and Truststores* section in the document *webMethods Integration Server Administrator's Guide*.

   The keystore must contain at least one private key, which can be used by Mediator as the signing alias.

   If Mediator is expected to verify the signature of incoming requests from clients, the keystore must also contain the public keys of the clients. The keystore should also contain the public key of the identity provider in its truststore, to validate the signature in the assertion which is signed by the identity provider. This will be useful for Holder-of-Key confirmation method-based requests.

2. In Integration Server, specify the keystore alias and signing alias to be used by Mediator, as described in the *Keystore, Truststore, and Key Aliases* section in the document *webMethods Integration Server Administrator's Guide*.

Configuring for SAML Holder-of-Key Processing

This section describes:

- "The Run-Time Processing of Holder-of-Key Tokens" on page 78

- "Configuring a Security Token Service (STS) for Holder-of-Key Processing" on page 78.
- "Configuring Integration Server, Mediator and Virtual Services for Holder-of-Key" on page 80.

The Run-Time Processing of Holder-of-Key Tokens

At run time, Mediator processes a request containing a Holder-of-Key token as follows:

1. The client sends a request for a SAML Token from a Security Token Service (STS).
2. The STS verifies/authenticates the client and creates a SAML assertion with key information that the client can use to sign the message when sending to the service provider.
3. The STS also signs the assertion with its private key to provide message integrity and non-repudiation.
4. The client receives the SAML assertion from the STS and creates a new SOAP request.
5. The client then adds the token in the SOAP WS-Security header and then signs the message with the same key information present in the SAML token to prove Proof-of-Possession of the token (thus acting as the Holder-of-Key).
6. The service receives the SOAP request with the SAML assertion and verifies that the SAML assertion was issued by a trusted STS.
7. The service also verifies that the message was signed by the same Subject specified in the SAML assertion, thus verifying that the client is the Holder-of-Key.
8. Once these conditions are satisfied, the service allows the request to proceed.

Configuring a Security Token Service (STS) for Holder-of-Key Processing

When determining which STS to use, consider the following:

- The STS must be able to provide a SAML 1.1 or 2.0 Holder-of-Key token to the client.
- The client must authenticate itself using X.509/Username/HTTP Token to the STS.
- STS issues a SAML assertion with the client's public key as the key information material in the token.
- The client uses its private key to sign the assertion before sending the request to Mediator.
- There are two freely available STS implementations:
  - Axis2.
  - JBoss PicketLink.

To configure an Axis2 STS for Holder-of-Key processing

1. Download Apache Axis2 1.5 and Rampart 1.5.
2. Run `policy\sample05` per the instructions in Rampart (ant service.05).
3. Make sure the service is deployed and accessible on the following link: `http://localhost:8080/axis2/services/STS?wsdl`.

   This Axis2 STS is now capable of issuing SAML 1.1 or 2.0 tokens.
4. Follow the instructions in `policy\sample05` to get the sample working if there are any problems.

Example

Notice that the services.xml file contains the description and configuration for the Axis2 STS, as follows:

```
<service name="STS">
  <module ref="rampart" />
  <module ref="addressing" />
  <module ref="rahas" />
  <parameter name="saml-issuer-config"
    <saml-issuer-config>
       <issuerName>SAMPLE_STS</issuerName>
       <issuerKeyAlias>sts</issuerKeyAlias>
       <issuerKeyPassword>apache</issuerKeyPassword>
         <cryptoProperties>
          <crypto provider="org.apache.ws.security.components.crypto.Merli
n">
            <property name="org.apache.ws.security.crypto.merlin.keystore.t
ype">
              JKS
            </property>
            <property name="org.apache.ws.security.crypto.merlin.file">
              sts.jks
            </property>
             <property name="org.apache.ws.security.crypto.merlin.keystore.
password">
              apache
            </property>
          </crypto>
         </cryptoProperties
         <timeToLive>300000</timeToLive>
```

Note that in the `services.xml` file the following modules are engaged:

- `rampart`, for security handling.
- `addressing`, for WS-Addressing header processing.
- `rahas`, for WS-Trust request/response processing.

The `saml-issuer-config` parameter specifies the configuration information for the STS, such as:

- `issuerName`: The name of this STS; will be used in the IssuerName element in the SAML Assertion.

- `timeToLive`: Specifies the duration of validity for the SAML Token.
- `issuerKeyAlias`: Refers to the private key in the specified keystore to use to sign the assertion.
- `issuerKeyPassword`: The password to access the private key in the keystore.

In addition, the `services.xml` file specifies the security requirements to access the STS through a WS-Security Policy, as shown below. The policy specifies that the client must sign the request body using an X.509 Token. Also note that the `RampartConfig` element specifies the configuration for the keystore to use by the STS.

```
.
.
.
<sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
<sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypol
icy">
  <sp:Body/>
</sp:SignedParts>
<ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
  <ramp:user>sts</ramp:user>
  <ramp:encryptionUser>client</ramp:encryptionUser>
    <ramp:passwordCallbackClass>
        com.softwareag.mediator.sts.PWCBHandler
    </ramp:passwordCallbackClass>
    <ramp:crypto provider="org.apache.ws.security.components.crypto.Merlin">
      <ramp:property:name="org.apache.ws.security.crypto.merlin.keystore.ty
pe">
        JKS
      </ramp:property>
      <ramp:property:name="org.apache.ws.security.crypto.merlin.file">
        sts.jks>sts.jks
      </ramp.property>
      <ramp:property:name="org.apache.ws.security.crypto.merlin.keystore.p
assword">
        apache
      </ramp:property>
.
.
.
```

Configuring Integration Server, Mediator and Virtual Services for Holder-of-Key

To configure Integration Server, Mediator and virtual services for Holder-of-Key

1. Ensure that you have created a keystore, as described in "Configuring Integration Server Keystores" on page 77.
2. Map the client certificate to a valid Integration Server user, as described in the *Certificate Mapping* section in the document *webMethods Integration Server Administrator's Guide*.

   Note: The value of the **Usage** field is irrelevant; the mapping of the client certificate only verifies that the request is signed by a valid Integration Server user.

3. Specify the Integration Server keystore in Mediator, as described in "Keystore Configuration" on page 70.
4. Create a truststore in Integration Server that holds the STS public key, as described in the *Keystores and Truststores* section in the document *webMethods Integration Server Administrator's Guide*.

   Note: Do not create a keystore for the STS. Only create a truststore.

5. Configure the list of trusted SAML token issuers in Integration Server as follows:
   a. Open the Integration Server Administrator if it is not already open.
   b. In the Navigation panel, select **Security > SAML**.
   c. Click **Add SAML Token Issuer**.
   d. Set the parameters as follows.

In this field...	Specify...
Issuer Name	The name of a SAML token issuer from which Integration Server should accept and process SAML assertions. This value must match the IssuerName value in the SAML token. If the SAML assertion issuer name does not match any configured issuers, the token will be rejected and a message similar to the following will be logged to the Server log: `2010-06-09 23:35:38 EDT [ISS.0012.0025E`
Truststore Alias	Text identifier for the truststore, which contains the public keys of the SAML token issuer.
Certificate Alias	Text identifier for the certificate associated with the truststore alias. This certificate alias must match the certificate used by the STS to sign the SAML assertion.
Clock Skew	The clock time difference (in milliseconds) between your Integration Server and the SAML token issuer.

   e. Click **Save Changes**.

   These parameter values are stored in the file *Integration Server_directory* \instances*instance_name* \config\security\saml \trusted_saml_issuers.cnf.

6. In CentraSite, include the "Require WSS SAML Token" action in the policies of your virtual services. Configure the action as described in the section *Run-Time Governance Reference* in the CentraSite documentation.
7. Client requests must meet the following requirements:
   - The request must contain a valid SAML Holder-of-Key token.
   - The request's SOAP body must be signed by the client using the private key corresponding to the public key present in the SAML assertion.

Configuring for SAML Sender-Vouches Processing

This section describes:

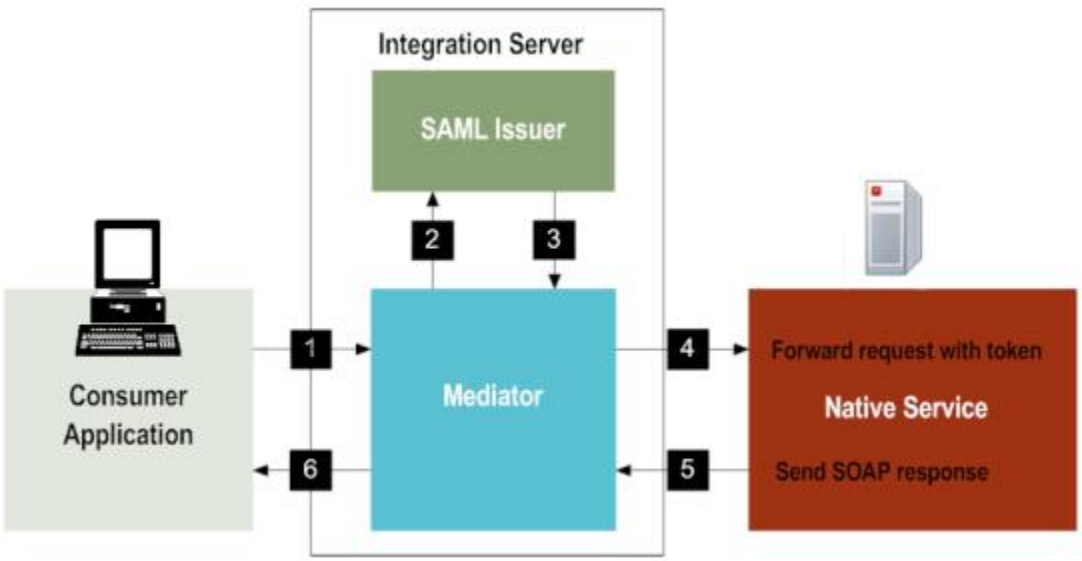
- "The Run-Time Processing of Holder-of-Key Tokens" on page 78
- "Configuring a Security Token Service (STS) for Holder-of-Key Processing" on page 78.
- "Configuring Virtual Services for Sender-Vouches Processing" on page 85.

The Run-Time Processing of Sender-Vouches Tokens

Mediator can act as a Security Token Service (STS) client. You can use Integration Server's default STS or you can use a third-party STS that has been defined in the Integration Server. The default STS supports only V2.0 SAML Sender-Vouches tokens.

The following illustration shows what happens at run time.

Mediator as an STS client



Step	Description
1	The user's client sends a SOAP request with SAML authentication information to Mediator. Integration Server authenticates the incoming request.
2	■ Mediator sends a WS-Trust RST to the STS to request a SAML v2 token. ■ Mediator sends the `<OnBehalfOf>` element that contains the authenticated user name to the STS.
3	The SAML Issuer sends the SAML v1/v2 assertion to Mediator.
4	Mediator forwards the SOAP request (along with the SAML assertion) to the native service. Mediator also uses the IS keystore and signing alias you specified to sign the SAML token and the request body before sending the request to the native service. Also, if you have configured the predefined Java service pub.mediator.security.ws.AddSamlSenderVouchesToken to add a timestamp in the outbound request, Mediator will sign the timestamp.
5	The native service sends a SOAP response to Mediator.
6	Mediator sends the response to the user's client.

Configuring a Security Token Service (STS) for Sender-Vouches Processing

To configure a Security Token Service (STS) for SAML Sender-Vouches processing

1. In Integration Server, create a keystore that will act as the keystore for the STS, as described in the *Keystores and Truststores* section in the document *webMethods Integration Server Administrator's Guide*.
2. Open the Integration Server Administrator if it is not already open.
3. In the Navigation panel, select **Solutions > Mediator > STS**.

   The **Security Token Service (STS) Configuration** page is displayed.
4. If you want to use Integration Server's default STS, select **DefaultSTS** and proceed to step 5.

   Note: If you want to use TcpMon to view the request/response, edit the default endpoint displayed on this page to point to the TcpMon port and forward the

request to the actual STS endpoint. This can be useful if Mediator sends back a SOAP fault to the client with an error about retrieving the SAML token.

5. Alternatively, you can use a third-party STS that has been defined in the Integration Server (as described in the *Web Services Developer's Guide*, in the section *Securing Web Services Using Policies Based on WS-SecurityPolicy*). To do this, click **Add new STS configuration** and set the parameters on the **Add Security Token Service (STS) Configuration** screen as follows.

For this parameter...	Specify...
Name	A unique name for the STS being configured. If this value is changed after creating an STS, the previous STS configuration will be deleted and replaced with the new one.
Endpoint	The STS endpoint to which the WS-Trust request will be sent by Mediator to obtain the SAML token.
Token Type	The type of token that Mediator should request from the STS. Value can be SAML_11 or SAML_20.
WS-Trust Version	The version of WS-Trust that Mediator should use to send the RST to the SAML Issuer. Value can be VERSION_05_02 or VERSION_05_12.
Time To Live (TTL)	Indicates the time-to-live value in seconds that will be specified in the RST. If not specified, the default is 300 seconds (5 minutes).
KeyStore / Signing Alias / Encryption Alias	Select a configured IS keystore. If the STS requires a signed and/or encrypted request, also specify the signing alias and the encryption alias.
HTTP Basic Authentication Username and Password	If the STS requires authentication, enter the HTTP Basic Auth username and password.
WS-Security Username	The WS-Security username token to send to the STS.
WS-Security Password	The password of the WS-Security username token.

For this parameter...	Specify...
WS-Security Password Type	The type of the password of the WS-Security username token.

6. If you selected Integration Server's default STS (**DefaultSTS**), edit the default STS's configuration file to specify the keystore and alias so that the STS can sign the SAML assertion it is issuing.

   The configuration file is:

   Integration Server_directory \instances*instance_name* \config\security\saml \esb_sts.xml

   The contents of the file are shown below. Use the comments as a guide to configure this file for your system.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- This configuration file is used to configure the IntegrationServer
token issuer that generates the SAML Sender Vouches token for Mediator
outbound requests -->

<IDataXMLCoder version="1.0">
<record javaclass="com.wm.data.ISMemDataImpl">

<!-- IssuerName - will be used as the IssuerName for each SAML token
issued by this Service; the default value is ESB_STS -->
<value name="IssuerName">ESB_STS</value>

<!-- IssuerKeystoreAlias - specify an Integration Server Keystore
Alias that contains the private keys that can be used to sign the
generated SAML Assertion -->
<value name="IssuerKeystoreAlias">STS</value>

   <!-- IssuerKeyAlias - the name of the key alias within the
IssuerKeystoreAlias that points to the private key files -->
<value name="IssuerKeyAlias">sts</value>

   <!-- TimeToLiveSeconds - how long in seconds the generated token
should be valid? the default is 300 seconds (i.e. 5 minutes) from the
time of token creation -->
<number name="TimeToLiveSeconds" type="java.lang.Integer">300</number>

   </record>
</IDataXMLCoder>
```

   After you edit the file, re-start Integration Server. **DefaultSTS** is now ready to issue SAML tokens.

Configuring Virtual Services for Sender-Vouches Processing

Next, you need to configure the desired virtual services so they can use the STS for Sender-Vouches processing.

To configure virtual services for SAML Sender-Vouches processing

1. Write an IS wrapper service that includes the predefined Java service mediator.security.ws:AddSamlSenderVouchesToken. This service will be called by Mediator during request processing.

   The value of this service's `ConfigName` parameter must be the STS you specified in "Configuring a Security Token Service (STS) for Sender-Vouches Processing" on page 83.

   For details about the AddSamlSenderVouchesToken service, see the CentraSite documentation (in the section *Virtualized Services in CentraSite Control* > *Invoking webMethods IS Services in Virtualized Services* > *Using the Security API in webMethods IS Services*).

2. In the Request Processing step of the desired virtual services, invoke the IS wrapper service you just created. For the procedure to do this, see the section *Virtualized Services in CentraSite Control* > *Configuring Virtual Services* in the CentraSite documentation.

   The virtual services are now ready to be deployed and invoked by the client.

3. Client requests must meet the following requirements:

   - The client must invoke the virtual services with valid Integration Server user credentials (HTTP Basic Authentication).
   - The credentials must be able to be used by Mediator to invoke the virtual services.
   - Mediator will use the identified Integration Server user as the value for the `<wst:OnBehalfOf>` element, shown below. If a virtual service's policy includes security actions such as the "Require WSS Username Token" action, the user identified by that token will be used as the value for `<wst:OnBehalfOf>` element when sending requests for SAML Sender-Vouches tokens.

```
<wst:OnBehalfOf>
    <wsse:UsernameToken xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity=utility- 1.0.xsd" wsu:Id="UsernameToken-28549389">
        <wsse:Username>Administrator</wsse:Username>
    </wsse:UsernameToken>
</wst:OnBehalfOf>
```

Configuring for SAML Bearer Token Processing

This section describes:

- "The Run-Time Processing of SAML Bearer Tokens" on page 87
- "Configuring a Security Token Service (STS) for Holder-of-Key Processing" on page 78
- "Configuring Integration Server, Mediator and Virtual Services for Bearer Tokens" on page 89

The Run-Time Processing of SAML Bearer Tokens

At run time, Mediator processes a request containing a SAML Bearer token as follows:

1. The client sends a request for a SAML Token from a Security Token Service (STS).
2. The STS verifies/authenticates the client and creates a SAML assertion that the client can send along with the message to the service provider.
3. The STS also signs the assertion with its private key to provide message integrity and non-repudiation.
4. The client receives the SAML assertion from the STS and creates a new SOAP request.
5. The client then adds the token in the SOAP WS-Security header.
6. The service receives the SOAP request with the SAML assertion and verifies that the SAML assertion was issued by a trusted STS.
7. Once these conditions are satisfied, the service allows the request to proceed.

Configuring a Security Token Service (STS) for SAML Bearer Token Processing

When determining which STS to use, consider the following:

- The STS must be able to provide a SAML 1.1 or 2.0 Holder-of-Key token to the client.
- STS issues a SAML assertion with the client's public key as the key information material in the token.
- There are two freely available STS implementations:
  - Axis2.
  - JBoss PicketLink.

To configure an Axis2 STS for Bearer token processing

1. Download Apache Axis2 1.5 and Rampart 1.5.
2. Run policy\sample05 per the instructions in Rampart (ant service.05).
3. Make sure the service is deployed and accessible on the following link: http://localhost:8080/axis2/services/STS?wsdl.

   This Axis2 STS is now capable of issuing SAML 1.1 or 2.0 tokens.
4. Follow the instructions in policy\sample05 to get the sample working if there are any problems.

Example

Notice that the services.xml file contains the description and configuration for the Axis2 STS, as follows:

```
<service name="STS">
```

```
  <module ref="rampart" />
  <module ref="addressing" />
  <module ref="rahas" />
  <parameter name="saml-issuer-config"
    <saml-issuer-config>
       <issuerName>SAMPLE_STS</issuerName>
       <issuerKeyAlias>sts</issuerKeyAlias>
       <issuerKeyPassword>apache</issuerKeyPassword>
         <cryptoProperties>
            <crypto provider="org.apache.ws.security.components.crypto.Merl
in">
               <property name="org.apache.ws.security.crypto.merlin.keystore
.type">
                 JKS
               </property>
               <property name="org.apache.ws.security.crypto.merlin.file">
                 sts.jks
               </property>
               <property name="org.apache.ws.security.crypto.merlin.keystore.
password">
                 apache
               </property>
            </crypto>
         </cryptoProperties
         <timeToLive>300000</timeToLive>
```

- `rampart`, for security handling.
- `addressing`, for WS-Addressing header processing.
- `rahas`, for WS-Trust request/response processing.

The `saml-issuer-config` parameter specifies the configuration information for the STS, such as:

- `issuerName`: The name of this STS; will be used in the IssuerName element in the SAML Assertion.
- `timeToLive`: Specifies the duration of validity for the SAML Token.
- `issuerKeyAlias`: Refers to the private key in the specified keystore to use to sign the assertion.
- `issuerKeyPassword`: The password to access the private key in the keystore.

In addition, the services.xml file specifies the security requirements to access the STS through a WS-Security Policy, as shown below. The policy specifies that the client must sign the request body using an X.509 Token. Also note that the `RampartConfig` element specifies the configuration for the keystore to use by the STS.

```
.
.
.
<sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
<sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypol
icy">
  <sp:Body/>
</sp:SignedParts>
<ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
  <ramp:user>sts</ramp:user>
  <ramp:encryptionUser>client</ramp:encryptionUser>
  <ramp:passwordCallbackClass>
```

```
       com.softwareag.mediator.sts.PWCBHandler
  </ramp:passwordCallbackClass>
    <ramp:crypto provider-"org.apache.ws.security.components.crypto.Merlin">
       <ramp:property:name="org.apache.ws.security.crypto.merlin.keystore.t
ype">
          JKS
       </ramp:property>
       <ramp:property:name="org.apache.ws.security.crypto.merlin.file">
          sts.jks>sts.jks
       </ramp.property>
       <ramp:property: name="org.apache.ws.security.crypto.merlin.keystore.p
assword">
          apache
       </ramp:property>
.
.
.
```

Configuring Integration Server, Mediator and Virtual Services for Bearer Tokens

Perform the same steps specified in "Configuring Integration Server, Mediator and Virtual Services for Holder-of-Key" on page 80, but with two differences:

- The request must contain a valid SAML Bearer token (instead of a Holder-of-Key token).
- The request's SOAP body *does not* have to be signed by the client using the private key corresponding to the public key present in the SAML assertion.

WS-Addressing Processing in Mediator

Mediator passes WS-Addressing from the client to the native service without changes. Using Mediator, clients can send WS-Addressing information to native services, as described in this section.

What is WS-Addressing?

WS-Addressing or Web Services Addressing is a specification of transport-neutral mechanisms that allow web services to communicate addressing information. It essentially consists of two parts: Message References and Message Information Headers.

- Message References

  An Endpoint Reference (EPR) is an XML structure encapsulating information useful for addressing a message to a Web service. It includes the destination address of the message, any additional parameters (called reference parameters) necessary to route the message to the destination, and optional metadata (such as WSDL or WS-Policy) about the service.

- Message Information Headers

Message Information Headers communicate addressing information relating to the delivery of a message to a Web service. The following properties are used in the message headers:

- Message destination URI (To)

  This URL is the same as the HTTP request's URL, but it is not required to be. Sample `To` header:

  ```
  <wsa:To> http://host/Service1</wsa:To>
  ```

- Source endpoint (From)

  This is the endpoint of the service that dispatched this message (EPR). `From` is an EPR of the message's sender. If the message's receiver needs to send a message back to the endpoint that sent the message, then it should use this EPR. It may be an Acknowledgement that needs to be sent back to the sender. Sample `From` header:

  ```
  <wsa:From>
    <wsa:Address> http://client/myClient</wsa:Address>
  </wsa:From>
  ```

- Reply endpoint (ReplyTo)

  This is the endpoint to which reply messages should be dispatched (EPR) any response from the web service should be sent to the ReplyTo EPR. Because `From` and `ReplyTo` can be two distinct EPRs, the message's sender might not be the endpoint that is meant to receive the response. Sample `ReplyTo` header:

  ```
  <wsa:ReplyTo>
    <wsa:Address> http://client/myReceiver</wsa:Address>
  </wsa:ReplyTo>
  ```

- Fault endpoint (FaultTo)

  This is the endpoint to which fault messages should be dispatched (EPR). If the response to a message is a SOAP fault, the fault should be sent to the EPR in the `FaultTo` header. Sample `FaultTo` header:

  ```
  <wsa:FaultTo>
    <wsa:Address>http://client/FaultCatcher</wsa:Address>
  </wsa:FaultTo>
  ```

- Action

  This is an action value indicating the semantics of the message URI (may assist with routing the message). Sample `Action` header:

  ```
  <wsa:Action> http://host/Operation1</wsa:Action>
  ```

- Unique message ID URI (MessageID)

  The `MessageID` header is nothing more than a URI that uniquely identifies a message. Sample `MessageID` header:

  ```
  <wsa:MessageID>uuid:098765</wsa:MessageID>
  ```

- Relationship to previous messages (a pair of URIs) (RelatesTo)

`RelatesTo` will typically be used on response messages to indicate that it is related to a previously known message and to define that relationship. The following sample `RelatesTo` header indicates that this is a Response message for a previously known Request message whose `MessageID` was `uuid:098765`. This header is critical in an asynchronous messaging scenario because the response message's receiver must be able to associate it with the original request message. Sample `RelatesTo` header:

```
<wsa:RelatesTo RelationshipType="wsa:Response">uuid:098765</
wsa:RelatesTo>
```

Implementation of WS-Addressing in CentraSite

CentraSite can import WSDLs that have WS-Addressing annotations.

CentraSite can virtualize WSDLs having WS-Addressing annotations. The annotations are kept in the WSDL for the virtualized service because they do not contain relevant information about the native service endpoint.

The following WS-Addressing headers can be included in a WSDL:

- `<wsam:UsingAddressing wsdl:required="true" />`
- Explicit `wsam:Action` attributes for operation input/output elements. For example:

  ```
  wsam:Action="http://greath.example.com/2004/wsdl/resSvc/
  opCheckAvailability"
  ```

- Default actions for inputs and outputs (for backwards compatibility with WSDL 1.1). The following pattern is used to construct a default action for inputs and outputs. The general form of an action IRI is as follows:

  wsam:Action="*[target namespace][delimiter][port type name][delimiter][input|output name]*"

- Endpoint references, such as:
  - `<wsa10:EndpointReference>`
  - `<wsa10:Address>`*`[url]`*`</wsa10:Address>`
  - `</Identity>`
  - `</wsa10:EndpointReference>`

What happens to these WS-Addressing headers?

- For a native service asset (created through import WSDL), the WSDL remains "as is"; no rewriting occurs.
- For a virtual service:
  - The WS-Policy attachment block (containing the `UsingAdressing` element) is removed.

- The `wsam:Action` attributes remain on the operations.
- The endpoint references are removed because they are replaced by Mediator's endpoint information.

WS-Addressing Scenarios for Mediator

The following scenarios for WS-Addressing are supported:

Scenario 1: Transparent Mode (Mediator Acts as a Proxy)

The client sends the SOAP message to Mediator, and Mediator acts as proxy. Mediator rewrites the WS-Addressing attributes, forwards to the message to the native service, rewrites the response, and sends the response back to the client.

Mediator handles the WS-Addressing elements as follows:

Element	Mediator Task	Comment
`<wsa:To>http://`*`host/service`*`</wsa:To>`	Replaces with native service address.	Address can be native service or Mediator address.
`<wsa:From>`	Does not modify.	Originates from the client and is not modified.
`<wsa:ReplyTo>`	Does not modify.	Originates from the client and is not modified.
`<wsa:FaultTo>`	Does not modify.	Originates from the client and is not modified.
`<wsa:Action>`	Does not modify.	Action is semantic information constructed from namespace and port information (by default), or the action consists of customer-specified information.
`<wsa:MessageID>`	Does not modify.	

Element	Mediator Task	Comment
`<wsa:RelatesTo>`	Does not modify.	

Implementation of WS-Addressing in Mediator

Mediator by itself does not support WS-Addressing as a proxy server. However, using Mediator, clients can send WS-Addressing information to the native service. There are two ways to achieve this functionality:

- "Method 1: Using an IS Flow Service for WS-Addressing" on page 93
- "Method 2: Client Request Sending WS-Addressing Information" on page 96

Method 1: Using an IS Flow Service for WS-Addressing

You can create an IS (Integration Server) flow service and configure it with public services provided by Mediator.

The run-time processing steps shown above are described below.

- 1. Client sends the request to Mediator without WS-Addressing headers.

  For example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://Username.Domain/service">
  <soapenv:Header/>
    <soapenv:Body>
      <ser:AddInts>
        <A>10</A>
        <B>34</B>
      </ser:AddInts>
    </soapenv:Body>
</soapenv:Envelope>
```

- 2. Mediator receives the request, appends the WSA header information through an IS flow service, and routes it to the native service endpoint.

  In the IS service we need to set WSA header information, such as the destination URI (`To`), `Action`, and `mustUnderstand`. The IS service will be configured in the request processing step in CentraSite.

The values of the WSA header information are:

- The destination URI (`To`) is `http://Username.Domain:5555/ws/service:AddInts_WSD/service_AddInts_WSD_Port`.
- `Action` is `service_AddInts_WSD_Binder_AddInts`.
- `mustUnderstand` is true.

For example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://Username.Domain/service">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:To soapenv:mustUnderstand="1">
      http://Username.Domain:5555/ws/service:AddInts_WSD/service_AddInts_
WSD_Port
    </wsa:To>
    <wsa:ReplyTo soapenv:mustUnderstand="1">
      <wsa:Address>
          http://www.w3.org/2005/08/addressing/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID soapenv:mustUnderstand="1">
      urn:uuid:a7ebc83f-6400-408e-abd1-2e9bf5187a46
    </wsa:MessageID>
    <wsa:Action soapenv:mustUnderstand="1">
      service_AddInts_WSD_Binder_AddInts
    </wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
    <ser:AddInts>
      <A>10</A>
      <B>34</B>
    </ser:AddInts>
  </soapenv:Body>
</soapenv:Envelope>
```

- 3. Native service receives the request from Mediator, processes it and sends a WSA response to Mediator.

  For example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <wsa:ReplyTo>
    <wsa:Address>
        http://www.w3.org/2005/08/addressing/anonymous
    </wsa:Address>
  </wsa:ReplyTo>
  <wsa:MessageID>
    urn:uuid:4bfd5406-060f-4180-aacf-0ba2f8a3d10e
  </wsa:MessageID>

  <wsa:Action>
     http://Username.Domain/service/AddInts_WSD_PortType/AddIntsResponse
  </wsa:Action>
  <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/
reply">
    urn:uuid:a7ebc83f-6400-408e-abd1-2e9bf5187a46
  </wsa:RelatesTo>
  </soapenv:Header>
```

```
  <soapenv:Body>
    <ser-root:AddIntsResponse xmlns:ser-root="http://Username.Domain/ser
vice" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <C>44</C>
    </ser-root:AddIntsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

- 4. Mediator receives the response from native service, removes WSA header information (shown in step 3), and sends the remaining information to the client.

  Since the client never sent the WS-Addressing header to the proxy server (Mediator), the WSA in the response is removed from the native service response. For example:

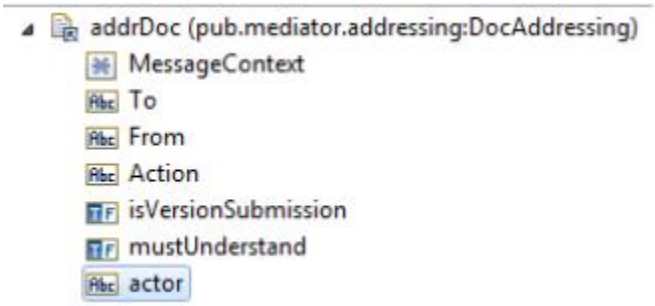
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ser-root:AddIntsResponse xmlns:ser-root="http://Username.Domain/ser
vice" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <C>44</C>
    </ser-root:AddIntsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Creating an IS Flow Service for WS-Addressing

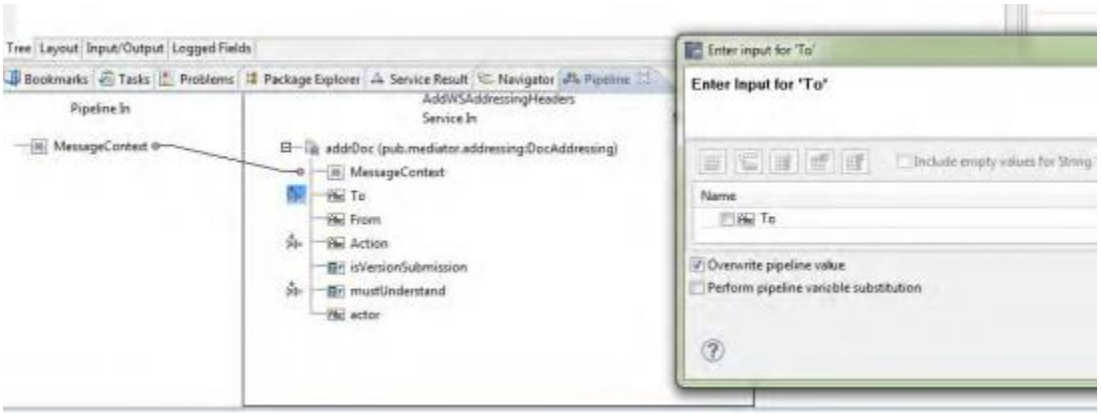
You need to create an IS flow service that sets the WSA headers. To accomplish this, you include in the flow service the Integration Server built-in service pub.mediator.addressing:AddWSAddressingHeaders.

To create an IS flow service for WS-Addressing

1. Create a flow service in Integration Server that has `MessageContext` as an input parameter.
2. This service should call the Integration Server built-in service pub.mediator.addressing:AddWSAddressingHeaders, which will accept the following doc type as input:



3. In the pipeline, map `MessageContext` and set the values for WSA headers as shown below.



4. Next, in the "Request Processing" step of the virtual service, invoke the IS flow service you just created.

Method 2: Client Request Sending WS-Addressing Information

The four run-time processing steps shown above are described below.

- 1. Client sends the request with WSA Header information to Mediator.

  In this example, the request has the WSA header information shown below.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://Username.Domain/service">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Action soapenv:mustUnderstand="1">
       service_AddInts_WSD_Binder_AddInts
    </wsa:Action>
    <wsa:MessageID soapenv:mustUnderstand="1">
       Test Message
    </wsa:MessageID>
    <wsa:To soapenv:mustUnderstand="1">
       http://127.0.0.1:2345/ws/Add_VS
    </wsa:To>
  </soapenv:Header>
  <soapenv:Body>
    <ser:AddInts>
      <A>25</A>
      <B>25</B>
    </ser:AddInts>
</soapenv:Body></soapenv:Envelope>
```

- 2. Since the client request contains the WSA header information, Mediator acts as a transparent proxy server.

Mediator receives the request and routes it to the native service endpoint without interfering; it does not process the WSA header information. For example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://Username.Domain/service">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Action soapenv:mustUnderstand="1">
       service_AddInts_WSD_Binder_AddInts
    </wsa:Action>
    <wsa:MessageID soapenv:mustUnderstand="1">
       Test Message
    </wsa:MessageID>
    <wsa:To soapenv:mustUnderstand="1">
       http://127.0.0.1:2345/ws/Add_VS
    </wsa:To>
  </soapenv:Header>
  <soapenv:Body>
    <ser:AddInts>
      <A>25</A>
      <B>25</B>
    </ser:AddInts>
</soapenv:Body></soapenv:Envelope>
```

- 3. Native service receives the request from Mediator, processes it, and sends a WSA response to Mediator.

  For example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://Username.Domain/service">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Action soapenv:mustUnderstand="1">
      service_AddInts_WSD_Binder_AddInts
    </wsa:Action>
    <wsa:MessageID soapenv:mustUnderstand="1">
      Test Message
    </wsa:MessageID>
    <wsa:To soapenv:mustUnderstand="1">
      http://127.0.0.1:2345/ws/Add_VS
    </wsa:To>
  </soapenv:Header>
  <soapenv:Body>
    <ser:AddInts>
      <A>25</A>
      <B>25</B>
    </ser:AddInts>
</soapenv:Body></soapenv:Envelope>
```

- 4. Mediator sends the response to the client with WSA header information.

  For example:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
       <wsa:MessageID>
          urn:uuid:3D1339DB6EBF867164293049269679 7-1700958707
       </wsa:MessageID>
       <wsa:Action>
            http://Username.Domain/service/AddInts_WSD_PortType/AddIntsR
esponse
       </wsa:Action>
       <wsa:RelatesTo>Test Message</wsa:RelatesTo>
```

```
    </soapenv:Header>
    <soapenv:Body>
       <ser-root:AddIntsResponse xmlns:ser-root="http://Username.Domain/
service" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <C>50</C>
       </ser-root:AddIntsResponse>
    </soapenv:Body>
   </soapenv:Envelope>
```

Note: The `pg.proxy.addressing.enabled` property should be set to false in `pg-config.properties` and should not be changed.

Mediator's GZIP Functionality

Mediator's GZIP functionality reduces the volume of data that is sent by native services' SOAP responses. Mediator can compress the responses based on the transport encoding (the Accept-Encoding and Content-Encoding HTTP headers).

Mediator will act as an intermediary, providing the capability of zipping and unzipping the response, even if the native service is not capable of doing so. Mediator compresses and uncompresses the responses depending on the Accept-Encoding and Content-Encoding HTTP headers from the client and the native service, respectively.

If:

- The client request contains an Accept-Encoding header set to `gzip` or variants such as:
  - `gzip,deflate`
  - `compress;q=0.5, gzip;q=1.0`
- Then the native service returns the response in either zipped or unzipped form (depending on how the native service's transport headers are configured) and sets the Content-Encoding field in the Response header to `gzip`.

For example:

Sample Request Header:

```
POST http://127.0.0.1:5555/ws/gzipTest
HTTP/1.1
Accept-Encoding: gzip,deflate|gzip|compress;q=0.5, gzip;q=1.0
Content-Type: text/xml;charset=UTF-8
```

Sample Response Header:

The native service will return the response in either zipped or unzipped form (depending on how the native service is configured), with the Content-Encoding field in the Response header set to `gzip` as follows:

```
HTTP/1.1 200 OK
WM_RESPONSE_TRANSPORT_HEADERS_MAP: Host=localhost:8080, Content-Length=393,
Accept-Encoding=gzip,deflate, SOAPAction="", User-Agent=Mozilla/4.0 [en]
(WinNT; I), Content-Type=text/xml; charset=UTF-8, Accept=image/gif, */*
Content-Encoding: gzip
```

```
Content-Type: text/xml; charset=utf-8Content-Length: 814
```

Mediator Behavior in Various GZIP Scenarios

The table below shows Mediator's behavior under various client and service configuration scenarios. These scenarios assume that the native service responds without any error.

Accept-Encoding Header Setting in Request	Content-Encoding Header Setting in Response	Original Content by Native Service	Mediator Behavior
gzip	not set	not encoded	Dynamically zips the response with Content-Encoding header set to gzip.
gzip,deflate compress;q=0.5, gzip;q=1.0	not set	not encoded	Dynamically zips the response with Content-Encoding header set to gzip.
* (any)	not set (or any encoding other than gzip)	not set (or any encoding other than gzip)	Passes the response to client (does not zip or unzip), with Content-Encoding set to the original content-encoding from the native service.
* (any)	gzip	encoded	Passes the zipped response to the client, with Content-Encoding set to gzip.
gzip	gzip	encoded	Passes the zipped response to client, with Content-Encoding set to gzip.
NULL (no encoding)	gzip	encoded	Unzips the content and removes the header, with Content-Encoding set to NULL.

Accept-Encoding Header Setting in Request	Content-Encoding Header Setting in Response	Original Content by Native Service	Mediator Behavior
NULL (no encoding)	null	not encoded	Passes the unzipped native service response to the client, with Content-Encoding set to NULL.

Mediator Response When Native Services Return Incorrect Content Encoding

In the above scenarios the native service responds with correct content encoding and correct content. But if the native service fails to do that (for example, if it zips the response but does not set the Content-Encoding error, or vice-versa), an Exception is thrown to the client and the error is logged. Such scenarios are shown in the table below.

Accept-Encoding Header Setting in Request	Content-Encoding Header Setting in Response	Original Content by Native Service	Mediator Response
gzip	not set	encoded	Throws Exception and logs error.
gzip	gzip	not encoded	Throws Exception and logs error.
gzip,deflate compress;q=0.5, gzip;q=1.0	gzip	not encoded	Throws Axis2 Exception and logs error.
* (any)	gzip	not encoded	Throws Axis2 Exception and logs error.
NULL	gzip	not encoded	Throws Axis2 Exception and logs error.

Mediator Response To Policy Violations

If Mediator rejects the request due to a policy violation, then only a plain response would reach the client, regardless of the Accept-Encoding.

Accept-Encoding Header Setting in Request	Content-Encoding Header Setting in Response	Original Content in Native Service	Mediator Response
gzip or gzip,deflate compress;q=0.5, gzip;q=1.0	any	any	1. Mediator rejects the request from the client and not the response from the native service. In this case, the request will not go to the native service. 2. Mediator sends an uncompressed response to the client.

Mediator Response When Native Services Return SOAP Faults

If the native service returns a SOAP fault, then Mediator will compress or decompress the response, based on the Accept-Encoding headers.

Accept-Encoding Header Setting in Request	Content-Encoding Header Setting in Response	Original Content in Native Service	Mediator Response
gzip or gzip,deflate compress;q=0.5, gzip;q=1.0	any	any	1. Mediator passes the request to the native service. 2. The server returns a SOAP fault. 3. Mediator compresses or uncompresses depending on the Accept-Encoding.

Mediator Behavior with Zipped Requests

Mediator also has the ability to process "gzipped" requests sent by the client:

- When the client sets the Content-Encoding:gzip and sends a zipped request to Mediator, then Mediator sends the zipped request to the native service with the Content-Encoding header unchanged.
- If the client sends to Mediator any Content-Encoding other then GZIP, then the header is removed and an uncompressed plain request is sent to the native service.

Configuring Custom Content-Types

By default, Mediator supports the following Content-Types:

- application/xml
- application/json and application/json/badgerfish
- text/xml
- multipart/form-data
- application/soap+xml
- application/x-www-form-urlencoded

In addition, you can define custom Content-Types for REST services and map them to the Content-Types listed above.

To do this, edit this file:

Integration Server_directory \instances*instance_name* \packages\WmMediator\config \resources\content-types.xml

which looks like this by default:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<content-types xmlns="http://contentTypes.mediator.softwareag">
    <!-- Please enter the custom content-types
    <content-type type="xml">
       <name></name>
    </content-type>
    -->
</content-types>
```

For example, if you want to define the custom Content-Types `myXMLContentType` and `myJsonContentType`, specify those names in the `<name>` tag as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<content-types xmlns="http://contentTypes.mediator.softwareag">
    <content-type type="xml">
       <name>myXMLContentType</name>
    </content-type>
    <content-type type="json">
       <name>myJsonContentType>/name>
    </content-type>
</content-types>
```

The `content-type type` attribute above identifies the `type` attribute of the Content-Type is to be mapped. The valid values are:

content-type type Value	Base Content-Type
text	text\xml, text\plain
xml	application\xml
soap	application\soap+xml
json	application\json
json-badgerfish	application\json\badgerfish
binary	application\octet-stream and other generic binary. MTOM attachments
url-encoded	application\x-www-form-urlencoded

After you finish defining the custom Content-Types, you must reload the WmMediator package. If a Content-Type with the same name already exits, the existing Content-Type will be overridden with the new one.

OAuth2 Inbound Configuration

You can configure your virtual services or virtual APIs to use the OAuth2 authentication scheme, as described in the following sections in the CentraSite documentation:

- *Virtualized Services in CentraSite Control* > *Configuring Virtual Services*.
- *API Management Solutions* > *Configuring the API Consumption Settings*.

The type of OAuth2 authorization grant that Mediator supports is "Client Credentials".

Client credentials are used as an authorization grant when the client is requesting access to protected resources based on an authorization previously arranged with the authorization server. That is, the client application gains authorization when it is registered with CentraSite.

You (the virtual service or API provider) must set the following parameters in Mediator to support OAuth2:

- Set `watt.server.auth.skipForMediator` to true (see "The watt.server.auth.skipForMediator Parameter" on page 104).
- Set the `pg.oauth2.` parameters as appropriate (see "The pg.oauth2 Parameters" on page 104).

Note: Mediator hosts a predefined service that consumers must invoke in order to receive OAuth2 access tokens (see "The Service for Obtaining OAuth2 Access Tokens" on page 105).

The watt.server.auth.skipForMediator Parameter

This parameter specifies whether Integration Server authenticates requests for Mediator. You must set this parameter 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, not just OAuth2) 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 watt.server.auth.skipForMediator

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.

The pg.oauth2 Parameters

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 following parameters in the Mediator properties file, which is located in:

Integration Server_directory \instances*instance_name* \packages\WmMediator\config \resources\pg-config.properties

- `pg.oauth2.isHTTPS`: Specifies the transport protocol over which the OAuth2 access tokens will be granted authorization. Set this parameter to true for HTTPS (the default) or false for HTTP

- `pg.oauth2.ports`: If pg.oauth2.isHTTPS is set to true, specify a comma-separated list of the HTTPS ports on which the service pub.mediator.oauth2.getOAuth2AccessToken will be available. For details about this service, see "The Service for Obtaining OAuth2 Access Tokens" on page 105.

The Service for Obtaining OAuth2 Access Tokens

Mediator hosts a REST service (pub.mediator.oauth2.getOAuth2AccessToken) to provide OAuth2 access tokens to consumers. Consumers can get access tokens by using the service URI and by specifying their client credentials. 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.

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
}
```

6 Configuring "SOAP Over JMS" Protocols

Overview

Mediator supports "SOAP over JMS" protocols (or "SOAP-JMS" for short) for sending messages between Web services. You can:

- **Create SOAP-JMS provider Web service endpoint aliases and SOAP-JMS triggers.**

  If a virtual service's "Entry Protocol Step" is set to JMS, a *provider Web service endpoint alias* is used to construct the endpoint reference, the JMS bindings, or both when consumers request the Web service. These endpoint aliases can be used by Mediator or JMS provider Web service descriptors to facilitate SOAP-JMS messaging. For more information, see "Creating a SOAP-JMS Provider Web Service Endpoint Alias and Trigger" on page 109.

  You must also create a SOAP-JMS trigger to receive and send SOAP-JMS messages.

- **Create SOAP-JMS consumer Web service endpoint aliases.**

  If a virtual service's "Routing Protocol" is set to JMS, a *consumer Web service endpoint alias* is used at run-time to generate a request and invoke an operation of the Web service. These endpoint aliases can be used by Mediator or JMS consumer Web service descriptors to facilitate SOAP-JMS messaging. For more information, see "Creating a SOAP-JMS Consumer Web Service Endpoint Alias" on page 116.

Because the behavior and configuration of SOAP-JMS protocols is similar to those of other JMS protocols in Integration Server, you should be familiar with its JMS configuration procedures. For more information about configuring JMS protocols in Integration Server, see *webMethods Integration Server Administrator's Guide*.

Configuring SOAP-JMS Messaging

Before you create a virtual service that uses JMS as its entry protocol or routing protocol, you must configure SOAP-JMS messaging on your system, as follows:

1. Configure and provide access to an external JMS service provider. You can use webMethods Broker to act as a JMS provider. Alternatively, you can use a third-party JMS provider or an open source JMS provider.
2. Use the JMS service provider to create request and response queues. For more information, see the documentation for your JMS service provider.
3. Specify the JMS client libraries in the Integration Server classpath so that Mediator is able to communicate with the JMS service provider.
4. Create one or more JNDI provider aliases to specify where Integration Server can look up administered objects when it needs create a connection to JMS provider or specify a destination for sending or receiving messages. For more information, see *webMethods Integration Server Administrator's Guide*.

5. Create one or more connection aliases that encapsulate the properties that Integration Server needs to create a connection with the JMS provider. For more information, see *webMethods Integration Server Administrator's Guide*.

   Tip: For this step, it is helpful to modify the default alias provided with Integration Server, `DEFAULT_IS_JMS_CONNECTION`, to work with your JMS service provider.

6. Configure a SOAP-JMS provider or consumer Web service endpoint alias in Integration Server. For more information, see "Creating SOAP-JMS Web Service Endpoint Aliases" on page 109.

Creating SOAP-JMS Web Service Endpoint Aliases

You might need to create one or both of the following kinds of endpoint aliases:

- A SOAP-JMS Provider Web Service Endpoint Alias (as described below).
- A SOAP-JMS Consumer Web Service Endpoint Alias (see "Creating a SOAP-JMS Consumer Web Service Endpoint Alias" on page 116).

Creating a SOAP-JMS Provider Web Service Endpoint Alias and Trigger

If a virtual service's "Entry Protocol Step" is set to JMS, you must create the following:

- A SOAP-JMS provider Web service endpoint alias.

  This alias stores the configuration used to generate the Web service's endpoint reference (the JMS URI), the JMS bindings, or both, in a WSDL file. The primary information used to build the JMS URI is retrieved from the SOAP-JMS trigger.

- A SOAP-JMS trigger.

  A trigger specifies the JMS provider destination from which the trigger will receive messages, and the name of the JMS connection alias that the trigger will use to connect to the JMS provider.

To create a SOAP-JMS provider Web service endpoint alias and a SOAP-JMS trigger

1. Open Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Settings > Web Services**.
3. Click **Create Web Service Endpoint Alias**.
4. Under **Web Service Endpoint Alias Properties**, provide the following information:

In this field...	Specify...
Alias	A name for the provider Web service endpoint alias.

In this field...	Specify...
	This will also become the name of the SOAP-JMS trigger. The alias name cannot include the following illegal characters: # ©\ & @ ^ ! % * : $. / \ \ ` ; , ~ + =) (| } {] [> < "
Description	A description for the endpoint alias.
Type	**Provider**
Transport Type	**JMS**

5. Under **JMS Transport Properties**, provide the following information:

In this field...	Specify...
SOAP-JMS Trigger Name	**WS Endpoint Trigger**. Integration Server populates this field automatically with all valid SOAP-JMS trigger names.
Delivery Mode (optional)	Whether the JMS transport should be **Persistent** or **Non_Persistent**.
Time to Live (optional)	The amount of time (in milliseconds) before a message expires. If set to zero, the message does not expire.
Priority (optional)	The message's priority in the queue. You can specify a number from 0 to 9, where 0 is the lowest priority and 9 is the highest.
Reply To Name (optional)	The name or lookup name of the JMS destination where a reply to the message should be sent.
Reply To Type (optional)	The type of message to which containing an empty value. The options are **Queue** and **Topic**. Default to empty value. **Note:** This parameter is only applicable when performing request-reply.
Connection Count	The maximum number of connections that can be used to retrieve messages for the JMS trigger. Specifying multiple connections per concurrent JMS trigger affects the maximum number of messages the JMS

In this field...	Specify...
	trigger can process. The **Max Execution Threads** property specifies how many messages each connection can process. The total number of messages is the product of **Max Execution Threads** and **Connection Count**. For example, if the JMS trigger has a **Max Execution Threads** value of 5 and **Connection Count** is set to 2, the JMS trigger can process up to 10 messages at one time. Restrictions: ■ The **Connection Count** property only applies to concurrent JMS triggers (that is, a JMS trigger in which the Processing mode is set to Concurrent). It does *not* apply to serial triggers. ■ The **Connection Count** property only applies if the JMS connection alias allows an individual connection for each JMS trigger (that is, if the **Create New Connection per Trigger** parameter is set to true/selected/yes). ■ The **Connection Count** property only applies if the JMS connection alias connects to the webMethods Broker. ■ The **Connection Count** property cannot be used if the JMS trigger receives messages from a non-durable subscriber for a topic.

6. Under **JMS WSDL Options**, provide the following information:

In this field...	Specify...
Include Connection Factory Name in JMS URI	Whether to include the connection factory name in the generated WSDL.
Include JNDI Parameters in JMS URI	Whether to include the JNDI parameters in the generated WSDL.

7. Under **WS Security Properties**, if the inbound SOAP request must be decrypted and/or the outbound SOAP request must be signed, do the following:

In this field...	Specify...
Keystore Alias	Alias of the keystore containing the private key used to decrypt the inbound SOAP request or sign the outbound SOAP response. **Important:** The provider must have already given the consumer the corresponding public key.
Key Alias	Alias of the private key used to decrypt the request or sign the response. The key must be in the keystore specified in **Keystore Alias**.

8. Under **WS Security Properties**, if the signing certificate chain of an inbound signed SOAP message has to be *validated*, specify the following:

In this field...	Specify...
Truststore Alias	The alias for the truststore that contains the list of CA certificates that Integration Server uses to validate the trust relationship.

9. Click **Save Changes.**

   Integration Server saves your changes and creates the SOAP-JMS trigger.

   For information about managing SOAP-JMS triggers, see the section below.

Viewing Thread Usage for SOAP-JMS Triggers

Integration Server Administrator displays the number of server threads currently used by each SOAP-JMS trigger on Integration Server.

To view thread usage for SOAP-JMS triggers

1. Open the Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the Navigation panel, click **Messaging**.
3. Click **JMS Trigger Management**.

   Under **Individual SOAP JMS Trigger Controls**, in the **Current Threads** column, Integration Server Administrator displays the number of server threads currently used to receive and process messages for each SOAP-JMS trigger. The **Current Threads** column displays **Not Connected** if Integration Server is not connected to a JMS provider.

Increasing or Decreasing Thread Usage for All Triggers

You can use Integration Server Administrator to limit the number of server threads that SOAP-JMS triggers can use. By default, Integration Server can consume up to 100% of the server thread pool for SOAP-JMS triggers. However, you should leave some server resources available to perform other functions.

Integration Server provides controls that you can use to manage server thread usage by all JMS triggers and SOAP-JMS triggers. You can use the controls to:

- Set the percentage of the server thread pool that Integration Server can use for receiving and processing all JMS and SOAP-JMS triggers.
- Reduce maximum execution threads by the same percentage across all concurrent JMS and SOAP-JMS triggers. This also decreases the rate at which concurrent JMS and SOAP-JMS triggers process messages.

To increase or decrease thread usage for all JMS and SOAP-JMS triggers

1. Open the Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the Navigation panel, click **Messaging**.
3. Click **JMS Trigger Management**.
4. Click **Edit JMS Global Trigger Controls**.
5. In the **Thread Pool Throttle** field, type the maximum percentage of the server thread pool that can be used for JMS and SOAP-JMS triggers. This includes threads used to retrieve messages from the JMS provider and threads used to process messages. You must enter a value greater than zero.
6. In the **Individual Trigger Processing Throttle** field, select the value, as a percentage of the configured maximum execution threads value, at which you want to the server to function. Integration Server applies this percentage to the maximum execution threads value for all concurrent JMS and SOAP-JMS triggers.

   This value applies to concurrent JMS and SOAP-JMS triggers only.
7. Click **Save Changes**.

Notes:

- If the **Thread Pool Throttle** percentage does not evaluate to a whole number when applied to the size of the server thread pool, Integration Server rounds up or down to the nearest whole number.
- Serial JMS and SOAP-JMS triggers always process one message at a time. For a serial trigger, Integration Server uses the same thread for receiving and processing a message. The **Individual Trigger Processing Throttle** does not affect serial JMS and SOAP-JMS triggers.
- Concurrent JMS and SOAP-JMS triggers use a pool of consumers to receive and process messages. Each consumer will use a thread from the server thread pool to receive and process a message. A concurrent JMS or SOAP-JMS trigger dedicates an

additional thread to managing the pool of consumers. For example, a concurrent JMS trigger configured to process up to 10 messages at a time can use a maximum of 11 server threads.

- If the percentage by which you reduce concurrent JMS or SOAP-JMS trigger execution threads does not resolve to a whole number for a JMS or SOAP-JMS trigger, Integration Server rounds up or rounds down to the nearest whole number. However, if rounding down would set the value to 0, the Integration Server rounds up to 1. For example, if you reduce **Individual Trigger Processing Throttle** to 10% of maximum, a concurrent JMS trigger with a maximum execution threads value of 12 would have an adjusted value of 1 (Integration Server rounds 1.2 down to 1). A concurrent JMS trigger with a maximum execution threads value of 4 would have an adjusted value of 1 (Integration Server rounds 0.4 up to 1).
- When you reduce the **Individual Trigger Processing Throttle** and save your changes, Integration Server does not meet the adjusted maximum by terminating any threads currently executing concurrent JMS and SOAP-JMS triggers. Integration Server enables server threads processing messages for concurrent JMS and SOAP-JMS triggers to execute to completion. Integration Server waits until the number of threads executing for a concurrent JMS or SOAP-JMS trigger is less than the adjusted maximum before dispatching another server thread for that JMS or SOAP-JMS trigger.

Enabling, Disabling, and Suspending SOAP-JMS Triggers

You can manage SOAP-JMS triggers and the amount of resources they consume by changing the state of a SOAP-JMS trigger. A SOAP-JMS trigger can have one of the following states:

Trigger State	Description
Enabled	The SOAP-JMS trigger is running and connected to the JMS provider. Integration Server retrieves and processes messages for the SOAP-JMS trigger.
Disabled	The SOAP-JMS trigger is stopped. Integration Server neither retrieves nor processes messages for the SOAP-JMS trigger.
Suspended	The SOAP-JMS trigger is running and connected to the JMS provider. Integration Server has stopped message retrieval, but continues processing any messages it has already retrieved.

Using the Integration Server Administrator, you can:

- Enable, disable, or suspend all SOAP-JMS triggers.
- Enable, disable, or suspend specific SOAP-JMS triggers.

You might want to change the state of all SOAP-JMS triggers at once as a quick way of making server resources available. This can be especially helpful in a situation in which Integration Server is functioning under heavy load and additional resources are needed immediately.

You might want to change the state for a specific SOAP-JMS trigger in the following situations:

- When a back-end system needs maintenance or is becoming unresponsive, you might want to suspend SOAP-JMS Triggers that interact with the back-end system. When the back-end system becomes available, you can enable the SOAP-JMS triggers.
- After suspending or disabling all SOAP-JMS triggers, you might enable specific SOAP-JMS triggers. For example, if the server is functioning under an unusually heavy load, you might first suspend all SOAP-JMS triggers and then gradually enable SOAP-JMS triggers, starting with the SOAP-JMS triggers involved in key processes.
- After Integration Server suspends a serial SOAP-JMS trigger automatically because a fatal error occurred during message processing.

The following procedure explains how to use Integration Server Administrator to change the state of all or individual SOAP-JMS triggers.

To enable, disable, or suspend SOAP-JMS triggers

1. Open the Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the Navigation panel, click **Messaging**.
3. Click **JMS Trigger Management**.
4. If you want to change the state of all SOAP-JMS triggers, do the following:
   a. Under **Individual SOAP JMS Trigger Controls**, in the **Enabled** column, click **edit all.**
   b. On the **Settings > Messaging > JMS Trigger Management > Edit State** screen, in the **New State** list, select the state that you want to apply to all SOAP-JMS triggers.
5. If you want to change the state of a specific SOAP-JMS trigger, do the following:
   a. Under **Individual SOAP JMS Trigger Controls**, in the row for the SOAP-JMS trigger that you want to enable, disable, or suspend, click the text in the **Enabled** column
   b. On the **Settings > Messaging > JMS Trigger Management > Edit State:***triggerName* screen, in the **New State** list, select the state that you want to apply to this SOAP-JMS trigger.
6. Click **Save Changes**.

Notes:

- If you want to disable a SOAP-JMS trigger, first suspend the SOAP-JMS trigger and wait for all the processing threads complete. Then disable the SOAP-JMS trigger. You can view the number of threads currently used by a SOAP-JMS trigger on the **Settings > Messaging > JMS Trigger Management** screen.

- When you disable a SOAP-JMS trigger, Integration Server does the following:
  - If the SOAP-JMS trigger is waiting before making a retry attempt, Integration Server interrupts processing for the SOAP-JMS trigger.
  - If the SOAP-JMS trigger is currently processing messages, Integration Server waits a specified amount of time before forcing the SOAP-JMS trigger to stop processing messages. If it does not complete in the allotted time, Integration Server stops the message consumer used to receive messages for the SOAP-JMS trigger and closes the JMS session. At this point the server thread for the SOAP-JMS trigger continues to run to completion. However, the SOAP-JMS trigger is not able to acknowledge the message when processing completes. If the delivery mode of the message is set to persistent, this can lead to duplicate messages.

    The time Integration Server waits between the request to disable the SOAP-JMS trigger and forcing the trigger to stop is specified by the watt.server.jms.trigger.stopRequestTimeout property.
- Because administered objects, such as destinations, are configured outside of Integration Server, disabling a SOAP-JMS trigger has no impact on the subscription.
- If a SOAP-JMS trigger is processing messages at the time it is suspended, the SOAP-JMS trigger will complete processing of those messages. The SOAP-JMS trigger also acknowledges the messages to the JMS provider.
- You can use the following built-in services to enable, disable, and suspend one or more triggers: pub.trigger:enableJMSTriggers, pub.trigger:disableJMSTriggers, and pub.trigger:suspendJMSTriggers. For details, see the *webMethods Integration Server Built-In Services Reference*.

Creating a SOAP-JMS Consumer Web Service Endpoint Alias

You must create a SOAP-JMS consumer Web service endpoint alias in either of the following cases:

- If the virtual service deployed to Mediator uses a JMS routing protocol.
- If the WSDL from the Web service provider does not supply either the JNDI provider or the JMS connection alias information.

There are two kinds of SOAP-JMS consumer Web service endpoint aliases you can create:

- One that includes a JNDI provider, as described below.
- One that includes a JMS connection alias (see "Creating a SOAP-JMS Consumer Web Service Endpoint Alias that Includes a JMS Connection Alias" on page 118).

Creating a SOAP-JMS Consumer Web Service Endpoint Alias that Includes a JNDI Provider

To create a SOAP-JMS consumer Web service endpoint alias that includes a JNDI provider

1. Open Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Settings > Web Services**.
3. Click **Create Web Service Endpoint Alias**.
4. Under **Web Service Endpoint Alias Properties**, provide the following information:

In this field...	Specify...
Alias	A name for the provider Web service endpoint alias. The alias name cannot include the following illegal characters: # ©\ & @ ^ ! % * : $. / \ \ ` ; , ~ + =) (| } {] [> < "
Description	A description for the endpoint alias.
Type	**Consumer**
Transport Type	**JMS**

5. Under **JMS Transport Properties**, provide the following information:

In this field...	Specify...
Connect Using	**JNDI Properties**
JNDI Provider Alias	The name of the JNDI provider alias.
Connection Factory Name	The connection factory JNDI lookup name.

6. Under **WS Security Properties**, provide the following information:

In this field...	Specify...
User Name	User name used to authenticate the consumer at the JMS transport level on the host server.

In this field...	Specify...
Password	The password used to authenticate the consumer on the host server.
Retype Password	Re-enter the above password.
Partner's Certificate	The path and file name of the provider's certificate, which contains its public key.
Keystore Alias	The alias of the keystore that contains the private key used to connect to the Web service host securely.
Key Alias	The alias of the key in the keystore that contains the private key used to connect to the Web service host securely. The key must be in the keystore specified in **Keystore Alias**.
Truststore Alias	The alias for the truststore that contains the list of CA certificates that Integration Server uses to validate the trust relationship.

7. Click **Save Changes**.

Creating a SOAP-JMS Consumer Web Service Endpoint Alias that Includes a JMS Connection Alias

To create a JMS consumer Web service endpoint alias that includes a JMS connection alias

1. Open Integration Server Administrator if it is not already open.
2. In the Navigation panel, select **Settings > Web Services**.
3. Click **Create Web Service Endpoint Alias**.
4. Under **Web Service Endpoint Alias Properties**, provide the following information:

In this field...	Specify...
Alias	A name for the provider Web service endpoint alias. The alias name cannot include the following illegal characters: # © \ & @ ^ ! % * : $. / \ \ ` ; , ~ + =) (| } {] [> < "
Description	A description for the endpoint alias.

In this field...	Specify...
Type	**Consumer**
Transport Type	**JMS**

5. Under **JMS Transport Properties**, provide the following information:

In this field...	Specify...
Connect Using	**JMS Connection Alias**
JMS Connection Alias	The name of the JMS connection alias.

6. Under **WS Security Properties**, provide the following information:

In this field...	Specify...
User Name	User name used to authenticate the consumer at the JMS transport level on the host server.
Password	The password used to authenticate the consumer on the host server.
Retype Password	Re-enter the above password.
Partner's Certificate	The path and file name of the provider's certificate, which contains its public key.
Keystore Alias	The alias of the keystore that contains the private key used to connect to the Web service host securely.
Key Alias	The alias of the key in the keystore that contains the private key used to connect to the Web service host securely. The key must be in the keystore specified in **Keystore Alias**.
Truststore Alias	The alias for the truststore that contains the list of CA certificates that Integration Server uses to validate the trust relationship.

7. Click **Save Changes**.

Built-In Services

The SOAP-JMS trigger uses the same built-in services as those used by the JMS trigger. For more information, see the *webMethods Integration Server Built-In Services Reference*.

A Advanced Settings

Introduction

This appendix describes the parameters you can specify in the Mediator properties file:

Integration Server_directory \instances*instance_name* \packages\WmMediator\config \resources\pg-config.properties

You can edit this file only by using a text editor. Before you edit the file, you should either:

- Shut down the Integration Server, make your changes, and restart the server,

  OR

- Make your changes and then reload the WmMediator package.

Note: Some parameters present in the `pg-config.properties` file can be edited from the Mediator Administration console in Integration Server Administrator. You do not have to restart Integration Server when you change parameters through the Mediator Administration console.

pg.3pSnmpSender.

pg.3pSnmpSender.sendDelay
This is an internal parameter. Do not modify.

pg.backupFailedProxies

pg.backupFailedProxies
Indicates whether the backup services failed. The default is `false`.

pg.CollectionPool.

pg.CollectionPool.minThreads
The minimum number of threads to be used for data collection (metrics and events). Specifying more threads means that Mediator can collect more data faster, but it will also increase the usage of system resources, which could result in slower service execution. The default is `1`.

pg.CollectionPool.maxThreads
The maximum number of threads to be used for data collection (metrics and events). This value must be greater than or equal to the value of `pg.CollectionPool.minThreads`. The default is `8`.

pg.CollectionPool.forcefulShutdown
Specifies whether the data collection thread pool should shut down immediately or wait for queued tasks to complete during Mediator shutdown. The default is `false`.

pg.CollectionPool.poolName
Sets the name for the data collection thread pool. The default is `CollectionPool`.

pg.CollectionWorkQueue.

pg.CollectionWorkQueue.queueCapacity
The size of the collection work queue to be used during data collection (metrics and events). This queue is used only when there are not enough collection pool threads to process all the data. For example, if `pg.CollectionPool.maxThreads` is set to 10, and the 10 threads are not sufficient for processing all the data, then the unprocessed data will be put into the collection work queue. If the queue capacity is reached, then any additional data will be lost. The default is `10000` items of data allowed in the queue.

Specifying a large queue and a small collection pool minimizes CPU usage and operating system resources, but this can lead to low throughput which will cause delays in data collection. Using a small collection work queue generally requires larger collection pool sizes, which keeps CPUs busier. This will avoid the delay but may encounter unacceptable scheduling overhead, which also decreases service execution.

pg.cs.snmpTarget.

These properties specify the settings for Mediator to use with the CentraSite SNMP server, which uses the SNMPv3 user-based security model.

Note: The default values are synchronized with the defaults used by CentraSite's SNMP Event Listener configuration file *CentraSite_directory* /cast/cswebapps/SOALinkSNMPEventsListener/WEB-INF/web.xml. The settings in both files must match.

pg.cs.snmpTarget.authProtocol
Specifies the authorization protocol to use for securing SNMPv3 messages. Valid values are `MD5` (default) and `SHA`.

You can edit this parameter from the **Authorization Protocol** field on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.cs.snmpTarget.base64Encoded
This is an internal parameter. Do not modify.

pg.cs.snmpTarget.connTimeout
Specifies the number of milliseconds before an inactive connection to the SNMP target server is closed. If set to 0, the socket remains open indefinitely.

pg.cs.snmpTarget.ipAddress
Specifies the IP address of the CentraSite SNMPv3 server.

You can edit this parameter from the **Host Name/IP Address** field on the **Mediator > Administration > SNMP** page in Integration Server Administrator. You cannot set this parameter to `localhost`.

> **Important:** If you do not set this parameter properly, the traps might not reach the SNMP server. Mediator will still send events as SNMP traps, but because there is no mechanism for acknowledging traps that are configured incorrectly, the SNMP server will not return errors when settings are incorrect.

pg.cs.snmpTarget.maxRequestSize
Specifies the maximum size (in bytes) for SNMP traps. The default is `10485760`.

pg.cs.snmpTarget.port
Specifies the SNMP trap receiver port on the CentraSite server that is listening for SNMP requests and packets. The default is `8181`.

You can edit this parameter from the **Port** field on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

> **Note:** 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.

pg.cs.snmpTarget.privProtocol
Specifies the privacy protocol to use for SNMPv3 messages. Valid values are `DES` (default), `AES128`, `AES192`, `AES256`, `3DES`, and `DESEDE`.

You can edit this parameter from the **Privacy Protocol** field on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.cs.snmpTarget.retries
Specifies the number of times to resend SNMP traps upon failure. The default is `1`.

This parameter works with `pg.cs.snmpTarget.sendTimeOut` to determine the delay in re-sending SNMP traps to malfunctioning SNMP servers (that is, it retries`*sendTimeOut`). This means that if the retries parameter is set to 3, and the `sendTimeOut` parameter is set to 500 milliseconds, there will be a 1.5 second delay before the Mediator thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes Mediator to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.cs.snmpTarget.sendTimeOut
Specifies the amount of time (in milliseconds) to wait before the SNMP trap times out because the server destination is not responding. This value schedules a timer that will resend an SNMP event that has not yet completed when it expires. You should set a timeout value that ensures that the trap is sent to the SNMP server within the time

frame. This parameter does not abort an event that is in progress. Set this parameter higher than the default when sending traps with large payloads. The default is `500`.

This parameter works with `pg.cs.snmpTarget.retries` to determine the delay in re-sending SNMP traps to malfunctioning SNMP servers (that is, it retries `*sendTimeOut`). This means that if the retries parameter is set to 3, and the `sendTimeOut` parameter is set to 500 milliseconds, there will be a 1.5 second delay before the Mediator thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes Mediator to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.cs.snmpTarget.sendTraps
Specifies whether to send traps to the CentraSite SNMPv3 server. The default is `false`.

You can edit this parameter from the **Send Traps to CentraSite** option on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.cs.snmpTarget.transportProtocol
Specifies the protocol used by SNMPv3 to send traps. Valid values are `TCP` (default) and `UDP`.

You can edit this parameter from the **Transport** field on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.cs.snmpTarget.useAuth
Specifies whether an authorization key is required. The default is `false`.

You can edit this parameter from the **Use Authorization** check box on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.cs.snmpTarget.usePrivacy
Specifies whether a privacy (encryption) key is required. The default is `false`.

You can edit this parameter from the **Use Privacy** check box on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.cs.snmpTarget.userName
Specifies the SNMPv3 user name to use when connecting to the receiver.

You can edit this parameter from the **User Name** field on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.csSnmpSender.

pg.csSnmpSender.sendDelay
This is an internal parameter. Do not modify.

pg.debug.

pg.debug.eventLoggerActive
This is an internal parameter. Do not modify.

pg.delayedRefresher.

Mediator cannot query CentraSite for updates or receive deployed services until Integration Server is running. If Integration Server is not yet fully operational when Mediator starts, a delayed refresh helper is used to wait for Integration Server. This helper will periodically check on Integration Server's status.

pg.delayedRefresher.napMillis
Specifies the amount of time (in milliseconds) the delayed refresher helper waits before checking to see whether Integration Server is running. The default is `500`.

pg.email.

pg.email.charset
Specifies the character set to use for the subject line, email addresses, and message body of the emails when sending alerts or events. The default is `US-ASCII`.

pg.email.debug
This is an internal parameter. Do not modify.

pg.email.from
Specifies the email address used when sending events by email. The default is `targetName@IS-hostname`.

You can edit this parameter from the **From** field on the **Mediator > Administration > Email** page in Integration Server Administrator.

pg.email.resourceMimeType
Specifies the MIME type Mediator uses to send the request and response payload attachments for transaction events that are sent by email. Mediator supports the following values:

- `application/gzip` (.gz)
- `application/zip` (.zip)
- `text/xml` (.xml)

The default is `application/gzip`.

pg.email.SenderActive
Specifies whether an email server is configured for Mediator. The default is `false`.

> **Note:** If a value is provided for the **SMTP Host Name/IP Address** field on the **Mediator > Administration > Email** page in Integration Server Administrator, this flag is set to true.

pg.email.smtpHost
Specifies the host name of the email server.

You can edit this parameter from the **SMTP Host Name/IP Address** field on the **Mediator > Administration > Email** page in Integration Server Administrator.

pg.email.smtpPort
Specifies the email port for the SMTP or SMTPS protocol. The default is `25`.

You can edit this parameter from the **Port** field on the **Mediator > Administration > Email** page in Integration Server Administrator.

pg.email.timeOut
Specifies the time out period (in milliseconds) when connecting to an e-mail server and sending e-mails. The default is `1000`.

pg.email.TLSEnabled
Specifies whether to use one-way transport-layer security (TLS). If set to true, the truststore configured for Mediator must include a certificate in the email server's certificate chain. The default is `false`.

You can edit this parameter from the **TLS Enabled** check box on the **Mediator > Administration > Email** page in Integration Server Administrator.

pg.email.userName
Specifies the user name of the email account used to log into the SMTP server.

You can edit this parameter from the **User** field on the **Mediator > Administration > Email** page in Integration Server Administrator.

pg.endpoint.

pg.endpoint.connectionTimeout
Specifies the time interval (in seconds) after which an HTTP connection attempt will timeout. Default: 30 seconds.

This is a global property that applies to the endpoints of all virtual services. If you prefer to specify a connection timeout for the endpoints of virtual services individually, set the **HTTP Connection Timeout** parameter in the virtual service's "Routing Protocols" processing step. This parameter takes precedence over `pg.endpoint.connectionTimeout`.

pg.endpoint.readTimeout
Specifies the time interval (in seconds) after which a socket read attempt will timeout. Default: 30 seconds.

This is a global property that applies to all virtual services. If you prefer to specify a read timeout for virtual services individually, set the **Read Timeout** parameter in the virtual service's "Routing Protocols" processing step. This parameter takes precedence over `pg.endpoint.readTimeout`.

pg.failedProxies.

pg.failedProxies.backupDir
The absolute or relative path to the `config` directory.

pg.http.

pg.http.ports
A comma-separated list of the HTTP ports on which Mediator and the deployed virtual services will be available.

You can edit this parameter from the **HTTP Ports Configuration** field on the **Mediator > Administration > General** page in Integration Server Administrator.

pg.https.ports
A comma-separated list of the HTTPS ports on which Mediator and the deployed virtual services will be available.

You can edit this parameter from the **HTTPS Ports Configuration** field on the **Mediator > Administration > General** page in Integration Server Administrator.

pg.IntervalPool.

The interval pool is used to schedule the processing of recurring tasks.

pg.IntervalPool.minThreads
Specifies the minimum thread count for this interval pool. The default is `1`.

pg.IntervalPool.maxThreads
Specifies the maximum thread count for this interval pool. The default is `1`.

pg.IntervalPool.forcefulShutdown
Specifies whether the interval thread pool should wait for queued tasks to complete during Mediator shutdown. Setting this parameter to true will cause Mediator to shut down immediately, without waiting for the tasks to finish. The default is `false`.

pg.IntervalPool.poolName
Specifies the name of the interval processor pool. The default is `IntervalPool`.

pg.jaxbFileStore.

pg.jaxbFileStore.consumerFileStore
Specifies the location of the locally persisted consumer applications that Mediator received from CentraSite. This file is updated periodically as long as communication with CentraSite is working. The default is `resources\consumers\consumers.xml`.

pg.jaxbNamesStore.

pg.jaxbNamesStore.namesFileStore
For internal use only. Specifies the locations of the consumer registered names store that Mediator received from CentraSite. This file is updated periodically as long as communication with CentraSite is working. The default is `resources\consumers\registeredNames.xml`.

pg.keystore.

pg.keystore.keyStoreHandle
This is an internal parameter. Do not modify.

pg.keystore.trustStoreHandle
This is an internal parameter. Do not modify.

pg.lb.

pg.lb.http.url
Specify the primary HTTP load balancer URL and port number to use in `http://`*hostname* `:`*portnumber* format.

You can edit this parameter from the **Load Balancer URL (HTTP)** field on the **Mediator > Administration > General** page in Integration Server Administrator.

pg.lb.https.url
Specify the primary HTTPS load balancer URL and port number to use in `http://`*hostname* `:`*portnumber* format.

You can edit this parameter from the **Load Balancer URL (HTTPS)** field on the **Mediator > Administration > General** page in Integration Server Administrator.

pg.lb.failoverOnDowntimeErrorOnly
Controls Mediator's behavior for load-balanced endpoints. The default is `false`, which means that in a load-balanced routing scenario, if a service fault is encountered in the response coming from endpoint 1, then Mediator will immediately try the next configured endpoint. There is no distinction on the type of fault present in the response from endpoint 1. However, if this parameter is set to `true`, then Mediator will failover only if the service fault is a "downtime" error (that is, if it matches one of the strings defined in the file:

Integration Server_directory \instances*instance_name* \packages\WmMediator\config \resources\downtime-patterns.txt

pg.nerv.

pg.nerv.jdbc.functional.pool.alias
Specifies the functional pool alias which is used to send run-time events and performance metrics using EDA.

pg.nerv.PgMenConfiguration.cache.size
Denotes the number of entries (events) that NERV will keep in the cache when trying to publish events to the EDA Destination. Default: 2000.

pg.nerv.PgMenConfiguration.publishLifeCycleEvents
Specifies whether to publish Lifecycle events to the EDA Destination. The default is false.

You can enable/disable this parameter from the **Lifecyle** check box on the **Mediator > Administration > EDA Configuration** page in Integration Server Administrator.

pg.nerv.PgMenConfiguration.publishErrorEvents
Specifies whether to publish Error events to the EDA Destination. The default is false.

You can enable/disable this parameter from the **Error** check box on the **Mediator > Administration > EDA Configuration** page in Integration Server Administrator.

pg.nerv.PgMenConfiguration.publishPolicyViolationEvents
Specifies whether to publish Policy Violation events to the EDA Destination. The default is false.

You can enable/disable this parameter from the **Policy Violation** check box on the **Mediator > Administration > EDA Configuration** page in Integration Server Administrator.

pg.nerv.PgMenConfiguration.perfDataEnabled
Specifies whether Mediator collects and reports performance data to the EDA Destination. The default is true.

> **Note:** If this parameter is disabled, Mediator removes all policy actions and will not trigger metrics reports.

You can enable/disable this parameter from the **Report Performance Data** check box on the **Mediator > Administration > EDA Configuration** page in Integration Server Administrator.

pg.nerv.PgMenConfiguration.reportInterval
Specifies the how often (in minutes) Mediator publishes performance data to the EDA Destination. The default is 60.

You can enable/disable this parameter from the **Publish Interval** field on the **Mediator > Administration > EDA Configuration** page in Integration Server Administrator.

pg.oauth2.

If your virtual services use the HTTP authentication scheme OAuth2, you should set these parameters.

pg.oauth2.isHTTPS
Specifies the transport protocol over which the OAuth2 access tokens will be granted authorization. Set this parameter to true for HTTPS (the default) or false for HTTP.

pg.oauth2.ports
If pg.oauth2.isHTTPS is set to true, specify a comma-separated list of the HTTPS ports on which the service pub.mediator.oauth2.getOAuth2AccessToken will be available. For details about this service, see "The Service for Obtaining OAuth2 Access Tokens" on page 105.

pg.passman.

pg.passman.configFile
This is an internal parameter. Do not modify.

pg.PgMenConfiguration.

pg.PgMenConfiguration.perfDataEnabled
Specifies whether Mediator collects and reports performance data to CentraSite. The default is `true`.

> **Note:** If this parameter is disabled, Mediator removes all policy actions and will not trigger metrics reports.

You can edit this parameter from the **Report Performance Data** check box on the **Mediator > Administration > CentraSite Communication** page in Integration Server Administrator.

pg.PgMenConfiguration.publishErrorEvents
Specifies whether to publish Error events to CentraSite. The default is `false`.

You can edit this parameter from the **Error** check box on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.PgMenConfiguration.publishLifeCycleEvents
Specifies whether to publish Life Cycle events to CentraSite. The default is `false`.

You can edit this parameter from the **Lifecycle** check box on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.PgMenConfiguration.publishPolicyViolationEvents
Specifies whether to publish Policy Violation events to CentraSite. The default is `false`.

You can edit this parameter from the **Policy Violation** check box on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.PgMenConfiguration.reportInterval
Specifies the how often (in minutes) Mediator publishes performance data to CentraSite. The default is `60`.

You can edit this parameter from the **Publish Interval** field on the **Mediator > Administration > CentraSite Communication** page in Integration Server Administrator.

pg.PgMenConfiguration.sieFlushInterval
Specifies the number of seconds before the accumulated invoked service events are pushed into the shared cache. The default is `2`.

pg.PgMenConfiguration.tickInterval
Specifies the amount of time (in seconds) between each interval processor iteration. This must be an evenly divisible fraction of the smallest policy interval, which is one minute. The default is `15`.

pg.PgMenSharedCacheManager.

pg.PgMenSharedCacheManager.lockTimeOut
Specifies the amount of time (in seconds) the shared cache waits to obtain a lock before timing out. The default is `5`.

pg.PgMetricsFormatter.

pg.PgMetricsFormatter.includeFaults
Specifies whether service faults are included in the aggregated performance metrics. If set to true, the average, minimum, and maximum response times will include failed requests in the calculations. The default is `false`. For more information, see "Key Performance Indicator Metrics and Run-Time Event Notifications" on page 21.

pg.policygateway.

pg.policygateway.targetName
Sets the name of the Mediator configured as a target in CentraSite. It is used by CentraSite to identify this instance of Mediator. The default is `your-target-name-here`.

You can edit this parameter from the **Target Name** field on the **Mediator > Administration > CentraSite Communication** page in Integration Server Administrator.

pg.policygateway.repositoryLocation
This is an internal parameter. Do not modify.

pg.policygateway.deleteTempArtifacts
Specifies whether to delete artifacts that are temporarily persisted by the deployment receiver. The default is `true`.

pg.proxyLoader

pg.proxyLoader.proxyLocation
This is an internal parameter. Do not modify.

pg.rampartdeploymenthandler.

pg.rampartdeploymenthandler.signingCertAlias
Specifies the signing alias used to sign the outgoing response from Mediator to the original request service consumer.

You can edit this parameter from the **Alias (Signing)** field on the **Mediator > Administration > General** page in Integration Server Administrator.

pg.rampartdeploymenthandler.usernameTokenUser
This is an internal parameter. Do not modify.

pg.ReportingPool.

Reporting pool options affect outbound event publishing. The pool includes all events, including key performance metrics (KPI) data.

pg.ReportingPool.minThreads
The minimum number of threads to be used for data reporting (metrics and events). Specifying more threads means that Mediator can send more events to the event destination faster, but it will also increase the usage of system resources, which could result in slower service execution. The default is `2`.

pg.ReportingPool.maxThreads
The maximum number of threads to be used for data reporting (metrics and events). This value must be greater than or equal to the value of `pg.ReportingPool.minThreads`. The default is `4`.

pg.ReportingPool.forcefulShutdown
Specifies whether the reporting pool should wait for queued tasks to complete during Mediator shutdown. Setting this parameter to true will cause Mediator to shut down immediately, without waiting for the tasks to finish. The default is `true`.

pg.ReportingPool.poolName
Specifies the name of the reporting pool. The default is `ReportingPool`.

pg.ReportingWorkQueue.

pg.ReportingWorkQueue.queueCapacity
The size of the reporting work queue to be used during data reporting (metrics and events). This queue is used only when there are not enough reporting pool threads to process all the data to be reported. For example, if `pg.ReportingPool.maxThreads` is set to 10, and the 10 threads are not sufficient for processing all the data, then the unprocessed data will be put into the reporting work queue. If the queue capacity is reached, then any additional data will be lost. The default is `5000` items of data allowed in the queue.

Specifying a large queue and a small reporting pool minimizes CPU usage and operating system resources, but this can lead to low throughput which will cause delays in data reporting. Using a small reporting work queue generally requires larger reporting pool sizes, which keeps CPUs busier. This will avoid the delay but may encounter unacceptable scheduling overhead, which also decreases service execution.

pg.serviceReader.

pg.serviceReader.interval
This is an internal parameter. Do not modify.

pg.snmp.communityTarget.

These parameters are used only when you have set `pg.snmp.customTarget` to `communityTarget` (see "pg.snmp.customTarget." on page 137).

Note: These settings must match those of your third-party SNMP server.

pg.snmp.communityTarget.base64Encoded
Specifies whether to use a third-party SNMPv1 community-based connection. If set to true, the community name entered in the **Community Name** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator should be the Base64 value. The default is `false`.

pg.snmp.communityTarget.communityName
Specifies the name used to interact with the SNMP receiver. This value must match the value that you set in the SNMP receiver. The default is `public`.

You can edit this parameter from the **3rd Party SNMP Configuration > Community Name** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.communityTarget.transportProtocol
Specifies the protocol used by SNMP to send traps. Valid values are `TCP` (default) and `UDP`.

If you select `UDP`, ensure that the SNMP server is in the same subnet, or else configure the routers to get packets across subnet boundaries. The maximum PDU size when running in UDP mode is 64Kb, which might be restrictive when sending large request and response payloads for Transaction Events.

You can edit this parameter from the **3rd Party SNMP Configuration > Transport** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.communityTarget.ipAddress
Specifies the IP address of the host running the SNMP server. You cannot set this parameter to `localhost`.

> **Important:** If you do not set this parameter properly, the traps might not reach the SNMP server. Mediator will still send events as SNMP traps, but because there is no mechanism for acknowledging traps that are configured incorrectly, the SNMP server will not return errors when settings are incorrect.

You can edit this parameter from the **3rd Party SNMP Configuration > Host Name/IP Address** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.communityTarget.port
Specifies the port accepting requests for the SNMP server. The default is `2162`.

You can edit this parameter from the **3rd Party SNMP Configuration > Port** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.communityTarget.retries
Specifies the number of times to resend SNMP traps upon failure. The default is `1`.

This parameter works with `pg.snmp.communityTarget.sendTimeOut` to determine the delay in re-sending SNMP traps to malfunctioning SNMP servers (that is, it retries `*sendTimeOut`). This means that if the retries parameter is set to 3, and the `sendTimeOut` parameter is set to 500 milliseconds, there will be a 1.5 second delay before the Mediator thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes Mediator to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.snmp.communityTarget.sendTimeOut
Specifies the amount of time (in milliseconds) to wait before the SNMP trap times out because the server destination is not responding. This value schedules a timer that will resend an SNMP event that has not yet completed when it expires. You should set a timeout value that ensures that the trap is sent to the SNMP server within the time frame. This parameter does not abort an event that is in progress. Set this parameter higher than the default when sending traps with large payloads. The default is `500`.

This parameter works with `pg.snmp.communityTarget.retries` to determine the delay in re-sending SNMP traps to non-responsive SNMP servers (that is, it retries `*sendTimeOut`). This means that if the retries parameter is set to 3, and the `sendTimeOut` parameter is set to 500 milliseconds, there will be a 1.5 second delay before the Mediator thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes Mediator to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.snmp.communityTarget.maxRequestSize
Specifies the maximum size (in bytes) for SNMP traps. The default is `65535`.

pg.snmp.customTarget.

Set these properties if you want to use a third-party SNMP server instead of the CentraSite SNMP server.

pg.snmp.customTarget
Specifies the security model of your third-party SNMP server:

- `communityTarget`, for the SNMPv1 community-based security model (the default). In addition, you need to set the parameters specified in "pg.snmp.communityTarget." on page 135.
- `userTarget`, for the SNMPv3 user-based security model. In addition, you need to set the parameters specified in "pg.snmp.userTarget." on page 137.

You can edit this parameter from the **SNMP Target Type** option on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.customTarget.connTimeout
Specifies the number milliseconds before an inactive connection to the third-party SNMP server is closed. If set to 0, the socket remains open indefinitely.

pg.snmp.customTarget.sendTraps
If set to `true`, Mediator will send traps to the third-party SNMP server. The default is `false`.

You can edit this parameter from the **Send Traps to 3rd Party SNMP Server** option on the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.userTarget.

These parameters are used only when you have set `pg.snmp.customTarget` to `userTarget` (see "pg.snmp.customTarget." on page 137).

Note: These settings must match those of your third-party SNMP server.

pg.snmp.userTarget.authProtocol
Specifies the authorization protocol to use. This parameter is valid only when `pg.snmp.userTarget.useAuth` is set to `true`. Valid values are `MD5` (default) or `SHA`.

You can edit this parameter from the **3rd Party SNMP Configuration > Authorization Protocol** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.userTarget.ipAddress
Specifies the IP address of the SNMP server. You cannot set this parameter to `localhost`.

Important: If you do not set this parameter properly, the traps might not reach the SNMP server. Mediator will still send events as SNMP traps, but because there is no mechanism for acknowledging traps that are configured incorrectly, the SNMP server will not return errors when settings are incorrect.

You can edit this parameter from the **3rd Party SNMP Configuration > Host Name/IP Address** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.userTarget.maxRequestSize
Specifies the maximum size (in bytes) for SNMP traps. The default is `65535`.

pg.snmp.userTarget.port
Specify the SNMP trap receiver port that is listening for SNMP requests and packets. The default is `2161`.

You can edit this parameter from the **3rd Party SNMP Configuration > Port** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.userTarget.privProtocol
Specifies the privacy protocol to use. This parameter is valid only when `pg.snmp.userTarget.usePrivacy` is set to `true`. The default is `DES`.

- `DES` (default)
- `AES128`
- `AES192`
- `AES256`

You can edit this parameter from the **3rd Party SNMP Configuration > Privacy Protocol** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.userTarget.retries
Specifies the number of times to resend SNMP traps upon failure. The default is `1`.

This parameter works with `pg.snmp.userTarget.sendTimeOut` to determine the delay in re-sending SNMP traps to malfunctioning SNMP servers (that is, it retries `*sendTimeOut`). This means that if the retries parameter is set to 3, and the `sendTimeOut` parameter is set to 500 milliseconds, there will be a 1.5 second delay before the Mediator thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes Mediator to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.snmp.userTarget.sendTimeOut
Specifies the amount of time (in milliseconds) to wait before the SNMP trap times out because the server destination is not responding. This value schedules a timer that will resend an SNMP event that has not yet completed when it expires. You should set a timeout value that ensures that the trap is sent to the SNMP server within the time frame. This parameter does not abort an event that is in progress. Set this parameter higher than the default when sending traps with large payloads. The default is `500`.

This parameter works with `pg.snmp.userTarget.retries` to determine the delay in re-sending SNMP traps to malfunctioning SNMP servers (that is, it retries `*sendTimeOut`).

This means that if the `retries` parameter is set to 3, and the `sendTimeOut` parameter is set to 500 milliseconds, there will be a 1.5 second delay before the Mediator thread sending the alert is available to send another event. Malfunctioning event destinations could delay the amount of time it takes Mediator to report events, or it could cause discarded events when the queue reaches its maximum level.

pg.snmp.userTarget.transportProtocol
Specifies the protocol used by SNMP to send traps. Valid values are `TCP` (default) and `UDP`.

You can edit this parameter from the **3rd Party SNMP Configuration > Transport** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.userTarget.useAuth
Specifies whether SNMP should support authenticated messages. Authenticated messages have a timestamp which ensures that if a user intercepts the request and then tries to send it repeatedly, the request is ignored.

If set to true, then the event is hashed to ensure that the contents are not modified by a third party while in transit. The default is `false`.

You can edit this parameter from the **3rd Party SNMP Configuration > Use Authorization** check box in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.userTarget.usePrivacy
Specifies whether a privacy (encryption) key is required. The default is `false`.

You can edit this parameter from the **3rd Party SNMP Configuration > Use Privacy** check box in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.snmp.userTarget.userName
Specifies the SNMPv3 user name to use when connecting to the receiver.

You can edit this parameter from the **3rd Party SNMP Configuration > User Name** field in the **Mediator > Administration > SNMP** page in Integration Server Administrator.

pg.vsdTransformer.

pg.vsdTransformer.xslFilePath
This is an internal parameter. Do not modify.

pg.uddiClient.

pg.uddiClient.hostName
Specifies the name or IP address of the machine on which CentraSite is running. The default is `your-CS-host-name`.

You can edit this parameter from the **Host Name** field on the **Mediator > Administration > CentraSite Communication** page in Integration Server Administrator.

pg.uddiClient.protocol
Specifies the protocol of the machine on which CentraSite is running. Valid values are `http` (the default) or `https`.

pg.uddiClient.uddiPort
Specifies the port used for UDDI access to CentraSite. The default is `53307`.

You can edit this parameter from the **UDDI Port** field on the **Mediator > Administration > CentraSite Communication** page in Integration Server Administrator.

pg.uddiClient.userName
Specifies the CentraSite user name and password that Mediator must use to access CentraSite. The default is `your-CS-user-name`.

If you are using the Operating System auth mechanism, use the following format for the user name:

`CS-Host-Name\CS-user-name`

You can edit this parameter from the **User Name** field on the **Mediator > Administration > CentraSite Communication** page in Integration Server Administrator.

pg.uddiClient.uddiClientTimeout
Specifies the number of milliseconds that can elapse before not publishing performance metrics to an unavailable CentraSite server. The default is `5000`.

B Server Configuration Parameters

Introduction

This appendix describes the Mediator-specific parameters you can specify in the server configuration file:

Integration Server_directory \instances*instance_name* \config\server.cnf

Typically you will use the **Settings > Extended** screen from the Integration Server Administrator to update this file, but there might be times when you need to edit the file directly using a text editor. If you edit the file directly, you should first shut down the Integration Server before updating the file. After you make your changes, restart the server. If you are using the **Settings > Extended** screen to update the server configuration file (server.cnf), server restart is not required unless otherwise specified.

watt.debug.

watt.debug.layout
Specifies the format of messages written to the server's log file and to the **Logs > Server** screen. For Mediator, you should set this parameter to the format "new", which will cause the Integration Server journal logging component to print a stack trace when an Error-level message is logged from the calling code sent in the exception. Messages in the "new" format will be of the following form:

(*Component*) [*ComponentID* .00*SubComponentID* .*SubComponentID* .*MessageKey*] *TimeStamp MessageType MessageText*

(IS.SERVER) [ISS.0025.25.6] 2007-07-31 10:45:27 EDT INFO: License Manager started

For more information about other watt.debug parameters, see *webMethods Integration Server Administrator's Guide*.

watt.net.

watt.net.maxClientKeepaliveConns
Sets the default number of client keep alive connections to retain for a given target endpoint. If not specified, five keep alive connections are retained. Setting this parameter may improve Mediator's performance when multiple threads are invoking a native service and all virtual service invocations are routing to the same endpoint.

watt.pg.

watt.pg.disableNtlmAuthHandler
If this property is set to false (the default), Mediator performs the NTLM Windows authentication if you select the HTTP Authentication mode "NTLM" in "Transparent" mode in the Routing Protocols processing step of a virtual service. If this property is

set to true, Mediator performs the Kerberos Windows authentication (and not NTLM Windows authentication).

watt.server.

watt.server.auth.skipForMediator

Specifies whether Integration Server authenticates requests from Mediator.

Any request to Mediator should not 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.

C Run-Time Events and Metrics Tables Details

Introduction

Refer to this appendix if you configured Mediator to use EDA to publish data about run-time events and key performance metrics (KPIs) to a database (as described in "EDA Configuration for Publishing Run-Time Events and Metrics" on page 57).

This appendix describes the published data for the following events and metrics:

Transaction Events Table

Table name: MED_EVENT_TXN

Column	Data Type	Description
EVENT_PK	NUMBER	Sequential primary key. This column will not contain a NULL value.
SESSION_ID	VARCHAR2(256)	Session token. This will be the IS session token or a GUID if the token is missing from the message context.
SERVICE_NAME	VARCHAR2(256)	Virtual service name. This column will not contain a NULL value.
OPERATION_NAME	VARCHAR2(256)	Virtual service operation (e.g. "invoke" for virtual service connector) hosting the invocation. This column will not contain a NULL value.
BINDING_NAME	VARCHAR2(256)	This column is currently not used.

Column	Data Type	Description
NATIVE_ENDPOINT	VARCHAR2(4000)	Native service endpoint to which the request has been routed.
TARGET_NAME	VARCHAR2(256)	The target name specified in the Integration Server Administrator's **Solutions > Mediator > Administration > CentraSite Communication** page.
CONSUMER_NAME	VARCHAR2(256)	The system-generated UUID. The UUID will be generated if an Evaluate action is present in the service's policy (assuming the Evaluate action's Identify User option is set). This column is null if no value is known.
CONSUMER_IP	VARCHAR2(64)	Consumer IP.
CONSUMER_ID	VARCHAR2(256)	The user-provided ID.

STATUS	VARCHAR2(256)	Formatted status based on resource message ID's.
		`0205.0033=SUCCESS`
		`0205.0034=ERROR {0}`
		where '{0}' is the fault reason.

RESPONSE	CLOB	CLOB with response data.
REQUEST	CLOB	CLOB with request data.
TOTAL_TIME	NUMBER	Time interval in milliseconds from when a request is received by the virtual service runtime until the response is returned to the caller. This field includes the overhead incurred while the message is passing through the Mediator runtime.
PROVIDER_TIME	NUMBER	Time interval in milliseconds from when Mediator forwards a request to the native provider until it receives the provider's response.

Column	Data Type	Description
		This field includes the time it takes a provider to process the request plus any network latency to/from the provider. This field is a better indicator of the time it takes for a provider to process a request and return a response. Subtracting total time from provider time should give a rough indicator of the Mediator overhead.
INSERTTIMESTAMP	TIMESTAMP(6)	Timestamp when the event is persisted to table.
AUDITTIMESTAMP	TIMESTAMP(6)	Timestamp when the event was created by the virtual service runtime.
ROOTCONTEXTID	CHAR(36)	This column is not used by EDA.
PARENTCONTEXTID	CHAR(36)	This column is not used by EDA.
CONTEXTID	CHAR(36)	This column is not used by EDA.
MSGID	CHAR(36)	This column is not used by EDA.
SERVERID	VARCHAR2(450)	This column is not used by EDA.
EVENT_CREATE_TS	TIMESTAMP(6)	Timestamp when the event was created by the virtual service runtime. This is not the time the database performed its insert (i.e., this is calculated by the Mediator policy engine and not a database function).
EVENT_USERNAME	VARCHAR2(256)	Integration Server always provides a value, even if anonymous. 80 is the size of user in IS_USER_TASKS table. This column will not contain a NULL value.

Column	Data Type	Description
EVENT_SOURCE	VARCHAR2(256)	Policy name that produced the event.
TOTAL_DATA_SIZE	NUMBER	Combined request and response payload sizes.

Monitoring Events Table

Table name: MED_EVENT_MON

Column	Data Type	Description
EVENT_PK	NUMBER	Sequential primary key. Constraint required.
SESSION_ID	VARCHAR2(256)	Session token. This will be the IS session token or a GUID if the token is missing from the message context. If this is a near real-time event, the session ID for the service invocation will be included in the event; however, if this is an aggregated event produced at the end of a policy's interval, no single service invocation is associated with the event, so no session is included.
SERVICE_NAME	VARCHAR2(256)	Virtual service name. Constraint required.
OPERATION_NAME	VARCHAR2(256)	Virtual service operation hosting the invocation. If this is a near real-time event, the virtual service operation name for the service invocation will be included in the event; however, if this is an aggregated event produced at the end of a policy's interval, no specific operation name is associated with the event. Constraint required.

Column	Data Type	Description
NATIVE_ENDPOINT	VARCHAR2(4000)	Native service endpoint to which the request has been routed.
BINDING_NAME	VARCHAR2(256)	This column is currently not used.
TARGET_NAME	VARCHAR2(256)	The target name specified in the Integration Server Administrator's **Solutions > Mediator > Administration > CentraSite Communication** page.
CONSUMER_NAME	VARCHAR2(256)	Depending on the action that reported the event ("Monitor Service Performance" or "Monitor Service Level Agreement") and the time the event was produced (near real-time or aggregate at interval end), zero or more consumer names could be resolved. As is the case with Mediator, if more than one consumer was included in the Monitor Service Level Agreement action definition, the complete list of consumers will be comma-delimited and is included for interval-end events. For a real-time event, the consumer associated with that event's service request will be included. So this implies that no consumers are ever specified for a Monitoring event produced by a Monitor action at interval's end.
CONSUMER_IP	VARCHAR2(64)	Consumer IP.
CONSUMER_ID	VARCHAR2(256)	The user-provided ID.
EVENT_CREATE_TS	TIMESTAMP(6)	Timestamp when the event was created by the virtual service runtime. This is not the time the database performed its insert (i.e., this is calculated by the Mediator policy engine and is not a database function).

Column	Data Type	Description
EVENT_SOURCE	VARCHAR2(256)	This column is not currently used.
ALERT_SOURCE	VARCHAR2(256)	The name of the policy that contains the "Monitor Service Performance" or "Monitor Service Level Agreement" action that produced the event.
ALERT_TYPE	VARCHAR2(256)	The value "Monitor" means the event was produced by the "Monitor Service Performance" action. The value "Sla" means the event was produced by the "Monitor Service Level Agreement" action or the "Throttling Traffic Optimization" action.
ALERT_DESC	VARCHAR2(256)	The alert message defined in the run-time policy action.
MONITOR_ATTR	VARCHAR2(256)	The expression rule that was breached in the run-time policy. If multiple conditions are specified for the policy, only the first expression is included here.
EVENT_USERNAME	VARCHAR2(256)	The Integration Server user that executed the service. If the user cannot be determined for the service request, Integration Server will use a default unprivileged user. So this field should never be NULL for near real-time events. However, if this is an aggregated event produced at the end of a policy's interval, no specific user name is associated with the event.

Policy Violation Events Table

Table name: MED_EVENT_PV

Column	Data Type	Description
EVENT_PK	NUMBER	Sequential primary key. Constraint required.
SESSION_ID	VARCHAR2(256)	Session token. This will be the IS session token or a GUID if the token is missing from the message context.
SERVICE_NAME	VARCHAR2(256)	Virtual service name. Constraint required.
OPERATION_NAME	VARCHAR2(256)	Virtual service operation hosting the invocation. If this is a near real-time event, the VS operation name for the service invocation will be included in the event; however, if this is an aggregated event produced at the end of a policy's interval, no specific operation name is associated with the event. Constraint required.
NATIVE_ENDPOINT	VARCHAR2(4000)	Native service endpoint to which the request has been routed.
BINDING_NAME	VARCHAR2(256)	This column is currently not used.
TARGET_NAME	VARCHAR2(256)	The target name specified in the Integration Server Administrator's **Solutions > Mediator > Administration > CentraSite Communication** page.
CONSUMER_NAME	VARCHAR2(256)	The system-generated UUID. The UUID will be generated if an Evaluate action is present in the service's policy (assuming the Evaluate action's Identify User option is set). This column is null if no value is known.
CONSUMER_IP	VARCHAR2(64)	Consumer IP.

Column	Data Type	Description
CONSUMER_ID	VARCHAR2(256)	The user-provided ID.
EVENT_CREATE_TS	TIMESTAMP(6)	Timestamp when the event was created by the virtual service runtime. This is not the time the database performed its insert (i.e., this is calculated by the Mediator policy engine and not a database function).
EVENT_SOURCE	VARCHAR2(256)	The Mediator policy name that produced the event.
ALERT_SOURCE	VARCHAR2(256)	The Mediator policy name that produced the event. Currently it is same as EVENT_SOURCE.
ALERT_TYPE	VARCHAR2(256)	PolicyViolation.
ALERT_DESC	VARCHAR2(256)	The alert message defined in the runtime policy action.
EVENT_USERNAME	VARCHAR2(256)	The Integration Server user that executed the service. If the user cannot be determined for the service request, Integration Server will use a default unprivileged user. So this field should never be NULL for near real-time events. However, if this is an aggregated event produced at the end of a policy's interval, no specific user name is associated with the event.

Error Events Table

Table name: MED_EVENT_ERR

Column	Data Type	Description
EVENT_PK	NUMBER	Sequential primary key. Constraint required.
SESSION_ID	VARCHAR2(256)	Session token. This will be the IS session token or a GUID if the token is missing from the message context.
SERVICE_NAME	VARCHAR2(256)	Virtual service name. Constraint required.
NATIVE_ENDPOINT	VARCHAR2(4000)	Native service endpoint to which the request has been routed.
BINDING_NAME	VARCHAR2(256)	This column is currently not used.
OPERATION_NAME	VARCHAR2(256)	Virtual service operation hosting the invocation. Constraint required.
TARGET_NAME	VARCHAR2(256)	The target name specified in the Integration Server Administrator's **Solutions > Mediator > Administration > CentraSite Communication** page.
CONSUMER_NAME	VARCHAR2(256)	The system-generated UUID. The UUID will be generated if an Evaluate action is present in the service's policy (assuming the Evaluate action's Identify User option is set). This column is null if no value is known.
CONSUMER_IP	VARCHAR2(64)	Consumer IP.
CONSUMER_ID	VARCHAR2(256)	The user-provided ID.
EVENT_CREATE_TS	TIMESTAMP(6)	Timestamp when the event was created by the virtual service runtime. This is not the time the database performed its insert (i.e., this is calculated by the Mediator

Column	Data Type	Description
		policy engine and not a database function).
EVENT_SOURCE	VARCHAR2(256)	The Mediator policy name that produced the event.
EVENT_USERNAME	VARCHAR2(256)	The Integration Server user used to execute the service. If the user cannot be determined for the service request, Integration Server will use a default unprivileged user. So this field should never be NULL for near real-time events. However, if this is an aggregated event produced at the end of a policy's interval, no specific user name is associated with the event.
ERROR_SOURCE	VARCHAR2(256)	The virtual service name producing this error event.
ERROR_DESC	VARCHAR2(4000)	The message from the underlying java.lang.Throwable that causes the error.

Lifecycle Events Table

Table name: MED_EVENT_LC

Column	Data Type	Description
EVENT_PK	NUMBER	Sequential primary key.
TARGET_NAME	VARCHAR2(256)	The target name specified in the Integration Server Administrator's **Solutions > Mediator > Administration > CentraSite Communication** page.
EVENT_CREATE_TS	TIMESTAMP(6)	Timestamp when the event was created by the virtual service runtime. This is not the time the database performed its insert (i.e.,

Column	Data Type	Description
		this is calculated by the Mediator policy engine and not a database function).
EVENT_STATUS	VARCHAR2(256)	Started or stopped.
EVENT_DESC	VARCHAR2(256)	Free text description including status.

Performance Metrics Table

Table name: MED_EVENT_METRICS

Column	Data Type	Description
EVENT_PK	NUMBER	Sequential primary key. Constraints required.
SERVICE_NAME	VARCHAR2(256)	Virtual service name. Constraints required.
OPERATION_NAME	VARCHAR2(256)	Virtual service operation hosting the invocation. Constraints required.
NATIVE_ENDPOINT	VARCHAR2(4000)	Native service endpoint to which the request has been routed.
BINDING_NAME	VARCHAR2(256)	This column is currently not used.
TARGET_NAME	VARCHAR2(256)	The target name specified in the Integration Server Administrator's **Solutions > Mediator > Administration > CentraSite Communication** page.
EVENT_CREATE_TS	TIMESTAMP(6)	Timestamp when the event was created by the virtual service runtime. This is not the time the database performed its insert (i.e., this is calculated by the Mediator policy engine and is not a database function).

Column	Data Type	Description
EVENT_SOURCE	VARCHAR2(256)	Policy name that produced the event.
INCLUDE_FAULTS	CHAR(1)	Boolean. This value impacts the min/max/avg response calculations since faults will be included in those metrics if this value is true.
INTERVAL_START	TIMESTAMP(6)	Start time of metrics collection interval.
INTERVAL_STOP	TIMESTAMP(6)	Stop time of metrics collection interval.
AVAIL	NUMBER	The percentage of time that a virtual service was available during the current interval.
LIVELY	CHAR(1)	Boolean. Specifies whether the service was considered available at the end of the current interval.
FAULT_COUNT	NUMBER	The number of failed service invocations during the current interval.
SUCCESS_COUNT	NUMBER	The number of successful service invocations during the current interval.
TOTAL_COUNT	NUMBER	The total number of service invocations during the current interval.
MAX_RESP	NUMBER	The maximum amount of time it took the service to complete an invocation in the current interval.
MIN_RESP	NUMBER	The minimum amount of time it took the service to complete an invocation in the current interval.

Column	Data Type	Description
AVG_RESP	NUMBER	The average amount of time it took the service to complete an invocation in the current interval.