CentraSite

Run-Time Governance Reference

Version 9.5 SP1

November 2013

This document applies to CentraSite Version 9.5 SP1.

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

Copyright © 2005-2013 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors..

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

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

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

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

Document ID: IINM-DG-ACTIONSR-95SP1-20140410

Table of Contents

Preface

This document describes the run-time events and performance metrics, as well as the run-time actions that you can apply to virtual services or virtual APIs.

The content is organized under the following sections:

Run-Time Events and Key Performance Indicator (KPI) Metrics	Describes: ■ The run-time events and Key Performance Indicator (KPI) metrics that can be collected and reported for each virtual service deployed in your system. ■ How to configure CentraSite to receive the events and metrics from the policy-enforcement point (such as Mediator) that collects them.
Built-In Run-Time Actions Reference for Virtual Services	You use these actions only when you are using CentraSite Control to create run-time policies for virtual services. This section provides: ■ A summary of the run-time actions. ■ An alphabetic reference of all actions and their parameters. ■ A listing of the action evaluation order and action dependencies. ■ Some common combinations of actions used to authenticate/identify consumers.
Built-In Run-Time Actions Reference for Virtual APIs	You use these actions only when you are using the CentraSite Business UI to create policy enforcement rules for virtual APIs. This section provides an alphabetic reference of all actions and their parameters.

1 Run-Time Events and Key Performance Indicator (KPI) Metrics

CentraSite can receive run-time events and Key Performance Indicator (KPI) metrics. A run-time event is an event that occurs while services are actively deployed on the 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 topics are discussed:

The Run-Time Event Types

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

Event Type	Description
Lifecycle	A Lifecycle event occurs each time Mediator is started or shut down.
Error	An Error event occurs each time an invocation of a virtual service results in an error.
Policy Violation	A Policy Violation event occurs each time an invocation of a virtual service violates a run-time policy that was set for the virtual service.
Transaction	A Transaction event occurs each time a virtual service is invoked (successfully or unsuccessfully).
Monitoring	Mediator publishes key performance indicator (KPI) metrics, such as the average response time, fault count, and availability of all virtual services (described below).

The Key Performance Indicator (KPI) Metrics

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

Metric	Reports on...
Availability	The percentage of time that a virtual service was available during the current interval. A value of 100 indicates that the service was always available. Only the time when the service is unavailable counts against this metric. If invocations fail due to policy violations, this parameter could still be as high as 100.
Average Response Time	The average amount of time it took the service to complete all invocations in the current interval. This is measured from the moment Mediator receives the request until the moment it returns the response to the caller.
Fault Count	The number of failed invocations in the current interval.
Maximum Response Time	The maximum amount of time it took the service to complete an invocation in the current interval.
Minimum Response Time	The minimum amount of time it took the service to complete an invocation in the current interval.
Successful Request Count	The number of successful service invocations in the current interval.
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 the section *Advanced Settings* in the document *Administering webMethods Mediator*.

The Event Notification Destinations

Mediator can publish all run-time events to an SNMP server. Mediator sends events as SNMP traps either to CentraSite's SNMP server or to a third-party SNMP server.

CentraSite's SNMP Server

CentraSite's SNMP server uses SNMPv3 user-security model.

Mediator delivers the webMethodsESB.MIB file to define the SNMP traps that it can produce. For the procedure to configure Mediator to send SNMP traps to the CentraSite SNMP server, see the section *SNMP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*. For more information about the webMethodsESB.MIB file, see the section *Run-Time Targets*.

Third-Party SNMP Servers

A third-party SNMP server can use either the SNMPv1 community-based security model or the SNMPv3 user-based security model.

To use a third party SNMP server, you must import or set up the MIB on all SNMP servers receiving SNMP traps from Mediator. For the procedure to configure Mediator to send SNMP traps to a third-party SNMP server, see the section *SNMP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.

Alerts and Transaction Logging

In addition to publishing all run-time events to an SNMP server, you can configure Mediator to:

- Send monitoring alerts to various destinations when user-specified performance conditions are violated.
- Log the payloads of all transactions to various destinations.

To do this, you include the following run-time actions in the policies of your virtual services. When you configure the run-time actions, you will specify the destinations for sending the alerts or logging the transactions.

To Send Monitoring Alerts...	Destination for Sending Alerts
Use any of the following monitoring actions: ■ "Monitor Service Performance" ■ "Monitor Service Level Agreement" ■ "Throttling Traffic Optimization"	■ The virtual service's Events profile in CentraSite. ■ Your Integration Server's local log. ■ An SMPT email server. ■ The CentraSite SNMP server or a third-party SNMP server.

To Log the Transaction Payloads	Destinations for Logging the Transactions
Use the "Log Invocations" action.	■ The virtual service's Events profile in CentraSite. ■ Your Integration Server's local log. ■ An SMPT email server. ■ Your Integration Server's audit log. ■ The CentraSite SNMP server or a third-party SNMP server.

The destinations for sending alerts and logging transactions are described below.

- SMTP Servers
- The Integration Server's Local Log
- The Integration Server's Audit Log

SMTP Servers

To specify an email destination, you must:

- Select the "Email" option as a destination when you configure the run-time actions listed above.
- Set the "Email Configuration" parameters in Integration Server Administrator (go to **Solutions > Mediator > Administration > Email**) as described in the section *SMTP Destinations for Alerts and Transaction Logging* in the document *Administering webMethods Mediator*.

The Integration Server's Local Log

To specify the Integration Server's local log as a destination, you must:

- Select the "Local Log" option as a destination when you configure the run-time actions listed above. When configuring the actions, you must also specify the severity of the messages to be logged (the logging level).
- Set the Integration Server Administrator's logging level for Mediator to match the logging levels specified for the run-time actions (go to **Settings > Logging > Server Logger**). For example, if a "Log Invocation" action is set to the logging level of Error, you must also set Integration Server Administrator's logging level for Mediator to Error. If the action's logging level is set to a low level (Warning-level or Informationlevel), but Integration Server Administrator's logging level for Mediator is set to a higher level (Error-level), then only the higher-level messages are written to the log file.

  Entries posted to the local log are identified by a product code of MED.

The Integration Server's Audit Log

You can select the Integration Server Audit Log as a destination for the “Log Invocation” action only. If you expect a high volume of invocations in your system, it is recommended that you select the Audit Log destination. For more information, see 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. For details, see the section *Configuring Communication with CentraSite* in the document *Administering webMethods Mediator*.

Mediator only tracks metrics for the current interval. At the end of the interval, Mediator aggregates the metrics and reports them to CentraSite. Once the metrics are reported, Mediator resets its counters for the new interval. Mediator does not calculate and aggregate metrics across intervals. If Mediator is shut down or the virtual service is undeployed before the current interval expires, the performance data is discarded.

Note: To avoid the need for Mediator to store metrics during periods of inactivity, Mediator stores only first and last zero value metrics that occurs during an interval, and discards the remaining consecutive zero value metrics. Doing this drastically reduces the storage space consumed by the metrics, and speeds the queries you perform in the dashboard. Skipping the in-between zero metrics will not affect in the performance graphs shown in the dashboard.

For more information about the metrics tracking interval, see the section *Key Performance Indicator Metrics and Run-Time Event Notifications* in the document *Administering webMethods Mediator*.

Configuring CentraSite to Receive Run-Time Events and Metrics

Prerequisites:

- Ensure that Mediator is configured for publishing events to an SNMP server, as described in the section *SNMP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.
- If you use a target type other than Mediator or webMethods Insight, be sure to configure CentraSite to publish events by providing your own MIB file in your target type's definition file, as described in the section *Run-Time Targets*. (CentraSite provides a MIB file for Mediator and Insight.)

- Optionally change CentraSite's default settings for logging run-time events, as described in the section *Logging*. By default, CentraSite logs all predefined event types, but you may disable any type.

CentraSite provides an Event Receiver, which is a data collector that collects the run-time event data. The Event Receiver listens for run-time events from the target instances via the SNMP (Application-Layer) protocol, and contains the logic to parse and store event data in the Event Receiver's data store. You must configure the Event Receiver's properties file as described below.

This section includes the following topics:

Components of the Event Receiver

The Event Receiver contains the following components.

- **The SNMP Listener**

  CentraSite's SNMPv3 Trap Listener, which supports **SNMP4J**. This Listener starts automatically when CentraSite starts.

- **The Intermediate Queue**

  The queue from the SNMP Listener to the Event Processor. This queue decouples the SNMP Listener threads from the Event Processor to improve throughput. The following modes are supported.

  - FileSystem: Incoming Traps will be stored temporarily in the file system
  - InMemory: Incoming Traps will be stored temporarily in memory
  - NoQueue: Incoming Traps will not be stored in any intermediate queue; the SNMP Listener threads will be processed.

  To select the mode, set the `eventsQueueImpl` property as described in ***Setting the Events Queue Implementation Property***.

- **The Event Processor**

  The Event Processor (SOALinkSNMPEventsListener) transforms incoming SNMPv3 Traps into an XML file (Events.xml) that complies with the schema in the RuntimeEvents Collection component. The Event Processor transforms an SNMPv3 Trap to the Events.xml file as follows:

  1. Determines the Event Type (and Target Type) to which the Trap belongs, and gets the corresponding UUIDs. This involves searching all Event Type-to-Trap mappings in all the defined

target types, using the Trap's OID. Since this is an expensive search, the Event Type-to-Trap mapping is cached to improve performance.

2. Parses the Trap attributes and obtains: the Service (UUID); the Target (Name); the TimeStamp and the SessionId. The Processor then searches the registry/repository and obtains the corresponding UUID for the Target Name. This mapping is also cached to improve performance.
3. Collects the remaining attributes from the Trap.
4. Constructs the Events.xml file using the Event Type UUID, Target Type UUID, Service UUID, Target UUID, TimeStamp, SessionId and other collected attributes.

- **The Batch Condition**

  The Batch Condition is a set of OR conditions used by the Event Processor. The Event Processor supports two modes of event storage into CentraSite: BatchMode and NoBatchMode. BatchMode is available only for FileSystem and InMemory queues. When BatchMode is enabled, the Event Processor continues to accumulate Events.xml documents until one of the conditions is evaluated as true. Then it inserts all the documents as a single batch into CentraSite.

  To specify BatchMode or NoBatchMode, set the batch-related properties as described in ***Setting the Properties for FileSystem or InMemory***.

- **The RuntimeEvents Collection**

  The run-time events are stored in the RuntimeEvents Collection as non-registry objects. For information about how events are stored, see ***Event Type Modeling***.

Configuring the Event Receiver

The Event Receiver is bundled in the installation as a Web-Application named SOALinkSNMPEventsListener supporting the JavaEE standard. The configuration file for the Event Receiver is located here:

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

The web.xml configuration file contains all the Event Receiver configuration properties. You must set these properties as described below, and then restart CentraSite.

- Setting the Database Configuration Properties
- Setting the SNMPv3 Transport Configuration Properties
- Setting the SNMPv3 USM Configuration Properties
- Setting the Events Queue Implementation Property

- Setting the Properties for FileSystem or InMemory

Setting the Database Configuration Properties

In the Event Receiver's configuration file, set the following properties related to the RuntimeEvents Collection database .

Database Property	Description
com.softwareag.centrasite.soalink.events.dbUrl	The URL of the RuntimeEvents Collection database. All run-time events will be persisted to this database.
com.softwareag.centrasite.soalink.events.dbUserId	The user name that the Events Listener will use for authentication before persisting event data to the RuntimeEvents Collection database. The default value of this property is the predefined user EventsUser. Optionally, you can change the value EventsUser to any login user who has the following privileges: - Write access on the Tamino collection "RuntimeEvents". - Read access on "TargetTypes", "Targets", "RuntimeEventTypes" and "LogUnit", which are under the Tamino collection "CentraSite". If you want to change the value to a login user, enter that login user's name in the form `<hostName>\<userName>`. **Important:** The predefined password of EventsUser is EventsManager4CS (there is no need to specify the password in this file). If you want to change this password, or if you have changed the value EventsUser to a login user, *you must change the password*. For details, see the section *Users, Groups, Roles, and Permissions*. Whenever you change the password, you must restart CentraSite.
com.softwareag.centrasite.soalink.events.dbNonActivityTimeOut	The non-activity timeout in seconds for the RuntimeEvents Collection database (default 2592000 seconds (i.e., 30 days)).

Setting the SNMPv3 Transport Configuration Properties

In the Event Receiver's configuration file, set the following properties related to a SNMPv3 Transport.

SNMPv3 Transport Property	Description
com.softwareag.centrasite.soalink.events.snmp.transport	Wire transport protocol that will be used by the SNMP Listener. Supported values are: TCP and UDP.
com.softwareag.centrasite.soalink.events.snmp.host	The CentraSite host name or IP address to which the SNMP listener will bind.
com.softwareag.centrasite.soalink.events.snmp.port	The port to which the SNMP listener will bind. The default is 8181. If Microsoft Internet Information Services (IIS) is installed (or will be installed) on the same machine hosting IS/Mediator, then you may want to change the default SNMP port of 8181 to something else, to avoid any potential runtime conflicts when sending SNMP packets.
com.softwareag.centrasite.soalink.events.snmp.maxInboundMessageSizeInBytes	Maximum inbound message size in bytes (an integer). Traps that exceed this size limit will be rejected. Default value is 256Kb.
com.softwareag.centrasite.soalink.events.snmp.dispatcherPoolSize	The SNMP Listener's Worker-Thread pool size (default is 10). This determines the throughput of the Listener.

Setting the SNMPv3 USM Configuration Properties

In the Event Receiver's configuration file, set the following properties related to SNMPv3 USM.

SNMPv3 USM Property	Description
com.softwareag.centrasite.soalink.events.snmp.engineId	EngineId to be used by the SNMP Listener. If the parameter is left blank, the SNMP Listener will auto-generate the engineId.
com.softwareag.centrasite.soalink.events.snmp.securityName	The SecurityName to be used by the SNMP Listener.
com.softwareag.centrasite.soalink.events.snmp.securityLevel	The Maximum SecurityLevel to be supported by SNMP Listener. Supported values in order are: NOAUTH_NOPRIV, AUTH_NOPRIV and AUTH_PRIV. For example, AUTH_PRIV provides the highest level of security but also supports the other two levels. Similarly AUTH_NOPRIV supports NOAUTH_NOPRIV.
com.softwareag.centrasite.soalink.events.snmp.authProtocol	AuthorizationProtocol to be used by the SNMP Listener for decoding the incoming trap. Supported values are: MD5 and SHA.
com.softwareag.centrasite.soalink.events.snmp.authPassPhraseKey	The PassPhrase key to be used by the AuthorizationProtocol. The passphrase key length should be >= 8. The key is stored in this file; the passphrase value is stored securely in passman.
com.softwareag.centrasite.soalink.events.snmp.privProtocol	The PrivacyProtocol to be used by the SNMP Listener for decoding the incoming trap. Supported values are: DES, AES128, AES, AES192, AES256, 3DES and DESEDE.
com.softwareag.centrasite.soalink.events.snmp.privPassPhraseKey	The PassPhrase key to be used by the PrivacyProtocol. The passphrase length should be >= 8. The key is stored in this file; the passphrase value is stored securely in passman.

Setting the Events Queue Implementation Property

In the Event Receiver's configuration file, set the following property related to the implementation of the events queue.

Events Queue Property	Description
com.softwareag.centrasite.soalink.events.eventsQueueImpl	Supported values are: - FileSystem: Incoming Traps will be stored temporarily in the file system - InMemory: Incoming Traps will be stored temporarily in memory - NoQueue: Incoming Traps will not be stored in any intermediate queue; the SNMP Listener threads will be processed one by one Additional, related properties are described in *Setting the Properties for FileSystem or InMemory*.

Setting the Properties for FileSystem or InMemory

When the *eventsQueueImpl* property is set to either FileSystem or InMemory, you should also set the following properties.

Property for FileSystem or InMemory	Description
com.softwareag.centrasite.soalink.events.enableBatchInsertion	Enable or disable batch insertion of events into the database. Supported values are true and false. If true, events will be batched as per the "batching rules" properties below, and the batch will be stored to the database. If false, events will be stored to the database one by one.
com.softwareag.centrasite.soalink.events.maxNumOfEventsPerBatch	Maximum number of events in a batch. Should be an integer value. A value <= 0 disables this rule. This rule is evaluated only on arrival of a new Trap.
com.softwareag.centrasite.soalink.events.maxSizeOfBatch	Maximum size (in bytes) of a batch. Default value is 512KB. Should be an integer value. A value <= 0 disables this rule. This rule is evaluated only on arrival of a new Trap.
com.softwareag.centrasite.soalink.events.maxTimeIntervalBetweenBatches	Maximum time interval (in milliseconds) between two subsequent batch storages. Should be an integer value. A value <= 0 disables this rule. Unlike the other two rules, this rule is evaluated periodically.

Property for FileSystem or InMemory	Description
	Hence this rule prevents any trap stuck in the batch for ever if inflow of traps stops; in short this acts as a batch-timeout. A very low value for this rule reduces batch efficiency and introduces unnecessary looping.
com.softwareag.centrasite.soalink.events.fileSystemQueueDir	(Only applies when the *eventsQueueImpl* property is set to FileSystem.) The directory that should be used as FileSystem Queue. Incoming traps will be stored in this directory temporarily and hence should have write permission. The path can be absolute or relative. It is advisable to provide the absolute path. Relative paths will be considered relative to one of the following, based on availability in the same order: 1. SOALinkSNMPEventsListener/WEB-INF directory for exploded deployments. 2. javax.servlet.context.tempdir for zipped deployments. 3. java.io.tmpdir if none of the above are available.

Event Type Modeling

Event types are modeled as registry objects. The String, Date, Integer and Boolean event attributes are stored in the registry/repository as slots. The File-Type attributes (representing payloads/binary-data) are stored as HasExternalLink associations.

For example, consider the predefined event type Transaction. If you go to the **Target Type** details page, you will see the Transaction event type attributes (which are obtained from the webMethodsESB.mib file) as follows:

Attribute Name	Object ID	Type
Service	1.3.6.1.4.1.1783.201.1.1.1	String
Target	1.3.6.1.4.1.1783.201.1.1.2	String
Timestamp	1.3.6.1.4.1.1783.201.1.1.3	Date
Consumer	1.3.6.1.4.1.1783.201.1.1.4	String
RequestStatus	1.3.6.1.4.1.1783.201.1.1.5	String
ResponsePayload	1.3.6.1.4.1.1783.201.1.1.6	File

Attribute Name	Object ID	Type
RequestPayload	1.3.6.1.4.1.1783.201.1.1.7	File
ProviderRoundTripTime	1.3.6.1.4.1.1783.201.1.1.8	Integer
TotalRoundTripTime	1.3.6.1.4.1.1783.201.1.1.9	Integer
SessionID	1.3.6.1.4.1.1783.201.1.1.16	String
ConsumerIP	1.3.6.1.4.1.1783.201.1.1.17	String

All of these attributes (except the File-Type attributes RequestPayload and ResponsePayload) are stored as registry object slots, as follows:

Slot Key	Slot Type	Slot Value (Attribute)
uddi_16d34470-9a92-11dd-9b43-e319c2a6593c	xs:string	Service
uddi_f18b5a40-9a91-11dd-b95e-b4758b17b88b	xs:string	Target
uddi_c798d3c0-9a91-11dd-889e-b999c87ba6b7	xs:datetime	TimeStamp
uddi_a7476ff0-a108-11dd-9c38-d8fd010529cc	xs:string	Consumer
uddi_a7476ff0-a108-11dd-9c38-eac6d60fc855	xs:string	RequestStatus
uddi_a7476ff0-a108-11dd-9c38-f3f84c6111f0	xs:integer	ProviderRoundTripTime
uddi_a7476ff0-a108-11dd-9c38-d02170b3aae3	xs:integer	TotalRoundTripTime
uddi_21b67010-9a92-11dd-926a-991c4c180c79	xs:string	SessionID
uddi_a7476ff0-a108-11dd-9c38-d34f346cb3d5	xs:string	ConsumerIP

The File-Type attributes ResponsePayload and RequestPayload are stored as HasExternalLink associations, as follows:

Association Key	Association Name (Attribute)
uddi:a747704b-a108-11dd-9c38-fde9d932116a	ResponsePayload
uddi:a745265b-a108-11dd-9c38-bf43eee17363	RequestPayload

The "Target Type to Event Type Association" Object

A target type (represented as a concept) is associated with an event type (represented as a registry object) by a "Target Type to Event Type Association" object, which defines the "UUID to MIB OID" mapping.

The following table shows the contents of a sample object that associates the target type webMethods Mediator with the event type Transaction. The table's columns are described below.

- Attribute: The Attribute column is not part of the object; it is included here simply for your reference.
- Slot Key: Contains the UUID, which is obtained from the event type registry object.
- Slot Type: Contains the slot type, which is obtained from the event type registry object.

- Slot Value: Contains the event type attribute's Object Identifier (OID), which is obtained from the MIB file.

Attribute	Slot Key (Event Type UUID)	Slot Type	Slot Value (Event Attribute OID)
Service	uddi_16d34470-9a92-11dd-9b43-e319c2a6593c	xs:string	1.3.6.1.4.1.1783.201.1.1.1
Target	uddi_f18b5a40-9a91-11dd-b95e-b4758b17b88b	xs:string	1.3.6.1.4.1.1783.201.1.1.2
TimeStamp	uddi_c798d3c0-9a91-11dd-889e-b999c87ba6b7	xs:datetime	1.3.6.1.4.1.1783.201.1.1.3
Consumer	uddi_a7476ff0-a108-11dd-9c38-d8fd010529cc	xs:string	1.3.6.1.4.1.1783.201.1.1.4
RequestStatus	uddi_a7476ff0-a108-11dd-9c38-eac6d60fc855	xs:string	1.3.6.1.4.1.1783.201.1.1.5
ResponsePayload	uddi_a747704b-a108-11dd-9c38-fde9d932116a	xs:anyURI	1.3.6.1.4.1.1783.201.1.1.6
RequestPayload	uddi_a745265b-a108-11dd-9c38-bf43eee17363	xs:anyURI	1.3.6.1.4.1.1783.201.1.1.7
ProviderRoundTripTime	uddi_a7476ff0-a108-11dd-9c38-f3f84c6111f0	xs:integer	1.3.6.1.4.1.1783.201.1.1.8
TotalRoundTripTime	uddi_a7476ff0-a108-11dd-9c38-d02170b3aae3	xs:integer	1.3.6.1.4.1.1783.201.1.1.9
SessionID	uddi_21b67010-9a92-11dd-926a-991c4c180c79	xs:string	1.3.6.1.4.1.1783.201.1.1.16
ConsumerIP	uddi_a7476ff0-a108-11dd-9c38-d34f346cb3d5	xs:string	1.3.6.1.4.1.1783.201.1.1.17

Event Modeling

An event is an instance of an event type. Events are modeled in a separate schema from the event type schema. CentraSite models events as non-registry objects (to avoid storing large amounts of unwanted event data in the registry/repository), and instead stores event data in a database collection within the Event Receiver. CentraSite maps events to their corresponding event types, using the event types' UUIDs. Similarly, events are mapped to target types, targets and services using UUIDs and the event type attributes.

The stored event data will contain:

- The event Trap ID (MIB OID).
- The event Trap value, which consists of:
  - The attribute key (MIB OID).
  - The attribute value.

The event data is stored in the Event Receiver as an "events" doctype.

If an event contains payloads (e.g., File-Type attributes such as ResponsePayload and RequestPayload), the payloads are stored in the Event Receiver as a "payloads" doctype, and will be referenced by the event stored under the "event" doctype, using ino:id. This is used to reduce de-serialization of the usually large payloads, and to improve performance of queries on the stored events.

Viewing Run-Time Events and Metrics

You can view the run-time events and metrics that occurred for:

- A particular target or all targets (see *Viewing Run-Time Events and Metrics for Targets*).
- Each virtual service (see *Viewing Run-Time Events and Metrics for Virtual Services*).

Viewing Run-Time Events and Metrics for Targets

Use the following procedure to view lists of run-time events for a particular target or for all targets.

If you are using the Mediator target, ensure that Mediator is configured to send event notifications to the destination(s) that are applicable for each event type. For details, see *SNMP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.

Note: You must have the permissions to manage targets, as described in the section *Run-Time Targets*.

To view a list of run-time events for targets

1. In CentraSite Control, go to **Operations > Events > Event List**.
2. Use the following fields to filter the event list you want to view:

In this field...	Specify...
Target Type	The type of the target whose events you want to view.
Target	The target whose events you want to view (or select **All** to view events of all targets).
Event Type	A particular event type, or select **All** to view all event types. For descriptions of the predefined event types, see the *The Run-Time Event Types*.
Service Type	Select **All** or **Virtual Service**. **Note:** CentraSite does not provide out-of-the-box policy-enforcement for web services.
Date Range	A range of dates from which to view the events.
Start Date	Alternatively, select the check box next to this field and click the calendar and select a starting date and time.
End Date	Click the calendar and select an ending date and time.

3. Click the **Search** button.
4. The generated event list displays the following information:

Field	Description
Date/Time	The date/time that the event occurred. Click this hyperlinked value to view the **Event Detail** page, which will contain the event's SOAP request or response name in the Attribute column. Click the hyperlinked request or response name to display the full SOAP request or response.
Session ID	(Read-only.) The session ID that generated the event.
Event Type	(Read-only.) The type of event (e.g., Monitoring, Policy Violation, Error, etc.).
Service Name	(Read-only.) The name of the service that caused the event.
Service Type	(Read-only.) The service’s type.
Target	(Read only.) The target on which the event occurred.
Target Type	(Read only.) The type of the target on which the event occurred.

Note: To view the list of attributes that are mapped for each event type, go to the target type's detail page (see the section *Run-Time Targets*).

Viewing Run-Time Events for Virtual Services

You can view the events and metrics for a virtual service in its Events profile and its Performance profile. For details, see the section *Virtual Services in CentraSite Control*.

Creating Custom Run-Time Events

CentraSite provides the predefined event types described in ***The Run-Time Event Types***. In addition, you can create custom run-time events that CentraSite will monitor.

Note: Prerequisite: You must have the Manage Runtime Event Types permission. By default, the predefined roles CentraSite Administrator and Operations Administrator include this permission. For more information about roles and permissions, see the section *Users, Groups, Roles, and Permissions*.

Important: To enable CentraSite to recognize custom event types, ensure that your MIB file (which is contained in your target type definition file) contains the SNMP Traps metadata and Object Identifiers for the custom events. For more information, see the section *Run-Time Targets*.

▶ To create custom event types

1 In CentraSite Control, go to **Operations > Events > Event Types** to display the **Event Types** page.

The page displays all the predefined event types (Monitoring, Policy Violation, Transaction, Error and Lifecycle) and any custom event types that have been defined.

2 To view the details of any event type, click its hyperlinked name.

The list of attributes for the event type is displayed. You can edit the attributes of custom event types, but not the predefined event types (see *Modifying Custom Run-Time Events*).

3 To create a custom event type, click the **Add Event Type** button. In the **Add/Edit Event Type** page specify a name and description for the event type. Event type names can contain any character (including spaces), and are not case-sensitive.

4 In the Event Type Attribute panel, the following default attributes are displayed. These attributes are required and cannot be deleted.

Attribute	Data Type
TimeStamp	Date
Target	String
Service	String
SessionID	String

To create additional attributes, perform the following steps:

1. Click the plus button at the bottom of the attribute list.
2. Specify a name in the **Name** column and a value in the **Data Type** column (Boolean, File, Date, Integer or String). Attribute names can contain any character (including spaces).
3. To add another attribute, click the plus button at the bottom of the list.
4. To delete an attribute, click the minus button for the attribute you want to delete.
5. Click **Save**.

Modifying Run-Time Events

To edit and delete custom event types, perform the following steps.

To modify a custom run-time event

1 In CentraSite Control, go to **Operations > Events > Event Types** to display the **Event Types** page.

The page displays all event types that have been defined.

2 To delete a custom event type, select the check box next to the event type and click the **Delete** button.

3 To edit the attributes of a custom event type, perform the following steps:

1. Click its hyperlinked name to display the **Add/Edit Event Type** page.
2. You can change the value of an attribute's data type, but not its name. Data types can be Boolean, File, Date, Integer or String.
3. To add another attribute, use the plus button at the bottom of the list.
4. To delete an attribute, click the minus button next to the attribute.
5. Click **Save**.

2

Built-In Run-Time Actions Reference for Virtual Services

This section describes the built-in run-time actions that you can include in run-time policies for virtual services. You use these actions only when you are using CentraSite Control to create run-time policies for virtual services. The content is organized under the following sections:

Summary of the Run-Time Actions for Virtual Services

You can include the following kinds of built-in run-time actions in the run-time policies for virtual services:

- WS-SecurityPolicy 1.2 Actions
- Monitoring Actions
- Additional Actions

WS-SecurityPolicy 1.2 Actions

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

Authentication Actions (WS-SecurityPolicy 1.2)

Mediator uses the following authentication actions to verify that the requests for virtual services contain a specified WS-Security element:

Require WSS Username Token	Uses WS-SecurityPolicy authentication to validate user names and passwords that are transmitted in the SOAP message header for the WSS Username token.
Require WSS X.509 Token	Identifies consumers based on a WSS X.509 token.
Require WSS SAML Token	Uses a WSS Security Assertion Markup Language (SAML) assertion token to validate service consumers.

XML Security Actions (WS-SecurityPolicy 1.2)

These actions provide confidentiality (through encryption) and integrity (through signatures) for request and response messages.

Require Signing	Requires that a request's XML element (which is represented by an XPath expression) be signed.
Require Encryption	Requires that a request's XML element (which is represented by an XPath expression) be encrypted.
Require SSL	Requires that requests be sent via SSL client certificates, and can be used by both SOAP and REST services.

Require Timestamps	Requires that timestamps be included in the request header. Mediator checks the timestamp value against the current time to ensure that the request is not an old message. This serves to protect your system against attempts at message tampering, such as replay attacks.

Monitoring Actions

Mediator provides the following run-time monitoring actions:

Monitor Service Performance	This action monitors a user-specified set of run-time performance conditions for a virtual service, and sends alerts to a specified destination when these performance conditions are violated.
Monitor Service Level Agreement	This action provides the same functionality as "Monitor Service Performance" but this action is different because it enables you to monitor a virtual service's run-time performance especially for particular consumer(s). You can configure this action to define a *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	(Not available in Mediator versions below 9.0.) This action limits the number of service invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated. You can use this action to avoid overloading the back-end services and their infrastructure, to limit specific consumers in terms of resource usage, 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, you can use this action alone to identify consumers only by host name or IP address.
Require HTTP Basic Authentication	This action uses HTTP basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header against the Integration Server's user account.
Authorize User	This action authorizes consumers against a list of users and/or a list of groups registered in the Integration Server on which Mediator is running. You use this action in conjunction with an authentication action "Require WSS Username Token", "Require WSS SAML Token" or "Require HTTP Basic Authentication".
Authorize Against Registered Consumers	This action authorizes consumer applications against all Application assets that are registered in CentraSite as consumers for the service.
Log Invocations	Logs request/response payloads to a destination you specify.
Validate Schema	Validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Action Evaluation Order and Dependencies

When you deploy a virtualized service, CentraSite automatically validates the service's run-time policy (or policies) to ensure that:

- Any action that appears in a single policy multiple times is allowed to appear multiple times.

  For those actions that can appear in a policy only once (for example, Identify Consumer), Mediator will choose only one, which might cause problems or unintended results.
- All action dependencies are properly met. That is, some actions must be used in conjunction with another particular action.

CentraSite will inform you of any violation, and you will need to correct the violations before deploying the service.

Effective Policies

When you deploy a virtual service to Mediator, CentraSite combines the actions specified within the service's run-time policy (or policies) that apply to the virtual service, and generates what is called the *effective policy* for the virtual service. For example, suppose your virtual service is within the scope of two run-time policies: one policy that performs a logging action and another policy that performs a security action. When you deploy the virtual service, CentraSite automatically combines the two policies into one effective policy. The effective policy, which contains both the logging action and the security action, is the policy that CentraSite actually deploys to Mediator with the virtual service.

When CentraSite generates the effective policy, it validates the resulting action list to ensure that it contains no conflicting or incompatible actions. If the list contains conflicts or inconsistencies, CentraSite resolves them according to Policy Resolution Rules. For example, an action list can include only one Identify Consumer action. If the resulting action list contains multiple Identify Consumer actions, CentraSite resolves the conflict by including only one of the actions (selected according to a set of internal rules) in the effective policy and omitting the others.

The effective policy that CentraSite produces for a virtual service is contained in an object called a *virtual service definition* (VSD). The VSD is given to Mediator when you deploy the virtual service. After you deploy a virtual service, you can view its VSD (and thus examine the effective policy that CentraSite generated for it) from the CentraSite user interface or from the Mediator user interface.

The following table shows:

- The order in which Mediator evaluates the actions.

- Action dependencies (that is, whether an action must be used in conjunction with another particular action).
- Whether an action can be included multiple times in a single policy. If an action cannot be included multiple times in a single policy, Mediator selects just one for the effective policy, which may cause problems or unintended results.

Evaluation Order	Action	Dependency	Can include multiple times in a policy?
1	**Require SSL**	None.	If multiple actions appear, and one of them has its Client Certificate Required parameter set to Yes, only one occurrence of the action appears in the effective policy.
2	**Require HTTP Basic Authentication**	In Mediator versions below 9.0: None. In Mediator version 9.0 and above: Identify Consumer.	No. Mediator includes only one action in the effective policy.
3	**Require WSS Username Token**	Identify Consumer action.	No. Mediator includes only one action in the effective policy.
4	**Require WSS X.509 Token**	Identify Consumer action.	No. Mediator includes only one action in the effective policy.
5	**Require WSS SAML Token**	None.	No. Mediator includes only one action in the effective policy.
6	**Require Signing**	Identify Consumer action.	Yes. Mediator generates a UNION of all Require Signing actions for the effective policy.
7	**Require Encryption**	Identify Consumer action.	Yes. Mediator generates a UNION of all Require Encryption actions for the effective policy.
8	**Require Timestamps**	Require SSL, Require Signing *and* Require Encryption.	No. Mediator includes only one action in the effective policy.
9	**Identify Consumer**	If Identify Consumer's identifier field is set to: ■ HTTP Authentication Token, the action Require HTTP Basic Authentication is also required. ■ WS-Security Authentication Token, the action Require WSS Username Token is also required. ■ Consumer Certificate, the actions Require WSS X.509	No. Mediator includes only one action in the effective policy.

Evaluation Order	Action	Dependency	Can include multiple times in a policy?
		Token or Require Signing are also required.	
10	**Authorize User**	Require HTTP Basic Authentication, Require WSS Username Token *or* Require WSS SAML Token.	No. Mediator includes only one action in the effective policy.
11	**Authorize Against Registered Consumers**	Identify Consumer action.	No. Mediator includes only one action in the effective policy.
12	**Validate Schema**	None.	If at least one occurrence of the action is configured to validate requests, and at least one occurrence of the action is configured to validate responses, then Mediator includes in the effective policy an action to validate both requests and responses. Otherwise, an action is chosen which validates only requests or only responses (depending on the value of the Validate SOAP Messages parameter of the action).
13	**Log Invocation**	None.	No. Mediator includes only one action in the effective policy.
14	**Monitor Service Performance**	None.	Yes. Mediator includes all Monitor Service Performance actions in the effective policy.
15	**Monitor Service Level Agreement**	Identify Consumer action.	Yes. Mediator includes all Monitor Service Level Agreement actions in the effective policy.
16	**Throttling Traffic Optimization**	Identify Consumer (if the Limit Traffic for Applications option is selected).	Yes. Mediator includes all Throttling Traffic Optimization actions in the effective policy.

Usage Cases for Identifying/Authenticating Consumers

When deciding which type of identifier to use to identify a consumer application, consider the following points:

- Whatever identifier you choose to identify a consumer application, it must be unique to the application. Identifiers that represent user names are often not suitable because the identified users might submit requests for multiple applications.

- Identifying applications by IP address or host name is often a suitable choice, however, it does create a dependency on the network infrastructure. If a consumer application moves to a new machine, or its IP address changes, you must update the identifiers in the application asset.
- Using X.509 certificates or a custom token that is extracted from the SOAP message itself (using an XPATH expression), is often the most trouble-free way to identify a consumer application.

Following are some common combinations of actions used to authenticate/identify consumers.

- **Scenario 1: Identify consumers by IP address or host name**
  - The simplest way to identify consumers is to use the Identify Consumer action and set its `Identify User Using` parameter to specify either a host name or an IP address (or a range of IP addresses).
- **Scenario 2: Authenticate consumers by HTTP authentication token**
  Use the following actions:
  - Identify Consumer action, and set its `Identify User Using` parameter to HTTP Authentication Token (to identify consumers using the token derived from the HTTP header).
  - Require HTTP Basic Authentication.
  - Additionally, you can use one or both of the following:
    - Authorize User action (to authorize a list of users and/or groups registered in the Integration Server on which Mediator is running).
    - Authorize Against Registered Consumers action (to authorize consumer applications against all Application assets registered as consumers for a service in CentraSite).
- **Scenario 3: Authenticate consumers by WS-Security authentication token**
  Use the following actions:
  - Identify Consumer action, and set its `Identify User Using` parameter to WS-Security Authentication Token (to identify consumers using the token derived from the WSS Header).
  - Require WSS Username Token action.
  - Additionally, you can use one or both of the following:
    - Authorize User action (to authorize a list of users and/or groups registered in the Integration Server on which Mediator is running).
    - Authorize Against Registered Consumers action (to authorize consumer applications against all Application assets registered as consumers for a service in CentraSite).
- **Scenario 4: Authenticate consumers by WSS X.509 token**
  - Identify Consumer action, and set its `Identify User Using` parameter to Consumer Certificate (to identify consumers using the WSS X.509 token).
  - Require WSS X.509 Token action
  - Require SSL action.

Run-Time Actions Reference for Virtual Services

This section describes the following built-in run-time actions that you can include in run-time policies for virtual services:

- Authorize Against Registered Consumers
- Authorize User
- Identify Consumer
- Log Invocation
- Monitor Service Performance
- Monitor Service Level Agreement
- Require Encryption
- Require HTTP Basic Authentication
- Require Signing
- Require SSL
- Require Timestamps
- Require WSS SAML Token
- Require WSS Username Token
- Require WSS X.509 Token
- Throttling Traffic Optimization
- Validate Schema

Authorize Against Registered Consumers

Note: Dependency requirement: A policy that includes this action must also include the **Identify Consumer** action. However, if the **Identify Consumer** action is set to identify users via the **HTTP Authentication Token** option, then "Authorize Against Registered Consumers" should not be included in the policy.

Authorizes consumer applications against all Application assets that are registered in CentraSite as consumers for the service.

Input Parameters

None.

Authorize User

Note: Dependency requirement: A policy that includes this action must also include *one* of the following: the **Require WSS SAML Token** action or the **Identify Consumer** action with one of the following options selected: “HTTP Authentication Token” or “WS-Security Authentication Token”.

Authorizes consumers against a list of users and/or a list of groups registered in the Integration Server on which Mediator is running.

Input Parameters

`Perform authorization against list of users`	*Boolean* Authorizes consumers against a list of users who are registered in the Integration Server on which Mediator is running. Specify one or more users in the fields below this option.
`Perform authorization against list of groups`	*Boolean* Authorizes consumers against a list of groups who are registered in the Integration Server on which Mediator is running. Specify one or more groups in the fields below this option.

Note: By default, both of the input parameters are selected. If you de-select one of these parameters, the fields showing the list of users (or groups) is not displayed.

Identify Consumer

Mediator uses this action to identify consumer applications based on the kind of consumer identifier (IP address, HTTP authorization token, etc.) you specify. Alternatively, this action provides an option to allow anonymous users to access the assets.

Input Parameters

`Anonymous Usage Allowed`	*Boolean* Specifies whether to allow all users to access the asset, without restriction.	
	Value	**Description**
	`False`	*Default.* Allows only the users specified in the `Identify User Using` parameter to access the assets.
	`True`	Allow all users to access the asset. In this case, do not configure the `Identify User Using` parameter.
`Identify User Using`	*String* Specifies the kind of consumer identifier that the action will use to identify consumer applications.	
	Value	**Description**

	`IP Address`	Identifies one or more consumer applications based on their originating IP addresses.
	`Host Name`	Identifies consumer applications based on a host name.
	`HTTP Authentication Token`	Uses HTTP Basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header. Mediator authorizes the credentials against the list of users registered in the Integration Server on which Mediator is running. This type of consumer authentication is referred to as "preemptive authentication". If you want to use "preemptive authentication", you should also include the action **Require HTTP Basic Authentication** in the policy. If you choose to omit "Require HTTP Basic Authentication", the client will be presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. This type of consumer authentication is referred to as "non-preemptive authentication". For more information, see **Require HTTP Basic Authentication**. **Note:** If you select the value `HTTP Authentication Token`, do not include the `Authorize Against Registered Consumers` action in the policy. This is an invalid combination.
	`WS-Security Authentication Token`	Validate user names and passwords that are transmitted in the SOAP message header in the WSS Username Token. If you select this value, you should also include the action **Require WSS Username Token** in the policy.
	`Custom Identification`	Validates consumer applications based on an XML element (represented by an XPath expression).
	`Consumer Certificate`	Identifies consumer applications based on information in a WSS X.509 certificate. If you select this value, you should also include the action **Require WSS X.509 Token** or the action **Require Signing** in the policy.
	`Client Certificate for SSL Connectivity`	Validates the client's certificate that the consumer application submits to the asset in CentraSite. The client certificate that is used to identify the consumer is supplied by the client to the Mediator during the SSL handshake over the transport layer. In order to identify consumers by transport-level certificates, the run-time communication between the client and the Mediator must be over HTTPS and the client must pass a valid certificate. To use this option, the following prerequisites must be met: ■ In Integration Server, create a keystore and truststore, as described in *Securing Communications with the Server* in the *webMethods Integration Server Administrator's Guide*. ■ In Integration Server, create an HTTPS port, as described in *Configuring Ports* in the *webMethods Integration Server Administrator's Guide*.

		- Configure Mediator by setting the IS Keystore and IS Truststore parameters, as described in *Configuring Mediator > Keystore Configuration* in the document *Administering webMethods Mediator*. - Configure Mediator by setting the HTTPS Ports Configuration parameter, as described in *Configuring Mediator > Ports Configuration* in the document *Administering webMethods Mediator*.

Log Invocation

Logs request/response payloads. You can specify the log destination and the logging frequency. This action also logs other information about the requests/responses, such as the service name, operation name, the Integration Server user, a timestamp, and the response time.

Note: You can include this action multiple times in a policy.

Input Parameters

`Log the Following Payloads`	*String* Optional. Specifies whether to log all request payloads, all response payloads, or both.	
	Value	**Description**
	`Request`	Log all request payloads.
	`Response`	Log all response payloads.
`Log Generation Frequency`	*String* Specifies how frequently to log the payload.	
	Value	**Description**
	`Always`	Log all requests and/or responses.
	`On Success`	Log only the successful responses and/or requests.
	`On Failure`	Log only the failed requests and/or responses.
`Send Data To`	*String* Specifies where to log the payload. **Important:** Ensure that Mediator is configured to log the payloads to the destination(s) you specify here. For details, see *Alerts and Transaction Logging* in the document *Administering webMethods Mediator*.	
	Value	**Description**

	`CentraSite`	Logs the payloads in the virtual service's Events profile in CentraSite. Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see the section *Configuring Communication with CentraSite* in the document *Administering webMethods Mediator*.
	`Local Log`	Logs the payloads in the server log of the Integration Server on which Mediator is running. Also choose a value in the `Log Level` field: ■ Info: Logs error-level, warning-level, and informational-level alerts. ■ Warn: Logs error-level and warning-level alerts. ■ Error: Logs only error-level alerts. **Important:** The Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to **Settings > Logging > Server Logger**).
	`SNMP`	Logs the payloads in CentraSite's SNMP server or a third-party SNMP server. Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > SNMP**). For the procedure, see the section *SNMP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.
	`Email`	Sends the payloads to an SMTP email server, which sends them to the email address(es) you specify here. Mediator sends the payloads as email attachments that are compressed using gzip data compression. To specify multiple addresses, use the ⊞ button to add rows. Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > Email**). For the procedure, see the section *SMTP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.
	`Audit Log`	Logs the payload to the Integration Server audit logger. For information, see the *webMethods Audit Logging Guide*. **Note:** If you expect a high volume of events in your system, it is recommended that you select the Audit Log destination for this action.

Monitor Service Performance

This action monitors a user-specified set of run-time performance conditions for a virtual service, and sends alerts to a specified destination when the performance conditions are violated. You can include this action multiple times in a single policy.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), Mediator sends an alert as soon as the performance condition is violated, without having to wait until the end of the metrics tracking interval. You can choose whether to send an alert only once during the interval, or every time the violation occurs during the interval. (Mediator will send another alert the next time a condition is violated during a subsequent interval.) For information about the metrics tracking interval, see *The Metrics Tracking Interval*.

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), Mediator aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

This action does not include metrics for failed invocations.

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see the section *Configuring Communication with CentraSite* in the document *Administering webMethods Mediator*.

Input Parameters

`Action Configuration parameters`	Specify one or more conditions to monitor. To do this, specify a metric, operator, and a value for each metric. To specify multiple conditions, use the ⊞ button to add multiple rows. If multiple parameters are used, they are connected by the AND operator.	
`Name`	*String Array* The metrics to monitor.	
	Value	**Description**
	`Availability`	Indicates whether the service was available to the specified consumers in the current interval.
	`Average Response Time`	The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment Mediator receives the request until the moment it returns the response to the caller.

Monitor Service Level Agreement

Note: Dependency requirement: A policy that includes this action must also include the **Identify Consumer** action.

This action is similar to the Monitor Service Performance action. Both actions can monitor the same set of run-time performance conditions for a virtual service, and then send alerts when the performance conditions are violated. This action is different because it enables you to monitor run-time performance for *one or more specified consumers*. You can include this action multiple times in a single policy.

You can configure this action to define a *Service Level Agreement (SLA)*, which is a set of conditions that defines the level of performance that a consumer should expect from a service. You can use this action to identify whether a service's threshold rules are met or exceeded. For example, you might define an agreement with a particular consumer that sends an alert to the consumer if responses are not sent within a certain maximum response time. You can configure SLAs for each virtual service/consumer application combination.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), Mediator sends an alert as soon as the performance condition is violated, without having to wait until the end of the metrics tracking interval. You can choose whether to send an alert only once during the interval, or every time the violation occurs during the interval. (Mediator will send another alert the next time a condition is violated during a subsequent interval.) For information about the metrics tracking interval, see ***The Metrics Tracking Interval***.

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), Mediator aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

This action does not include metrics for failed invocations.

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see the section *Configuring Communication with CentraSite* in the document *Administering webMethods Mediator*.

Input Parameters

<table>
<tr><td>Action Configuration parameters</td><td colspan="2">Specify one or more conditions to monitor. To do this, specify a metric, operator, and value for each metric. To specify multiple conditions, use the ⊞ button to add multiple rows. If multiple parameters are used, they are connected by the AND operator.</td></tr>
<tr><td rowspan="2">Name</td><td colspan="2"><i>String Array</i> The metrics to monitor.</td></tr>
<tr><td><b>Value</b></td><td><b>Description</b></td></tr>
</table>

	`Availability`	Indicates whether the service was available to the specified consumers in the current interval.
	`Average Response Time`	The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment Mediator receives the request until the moment it returns the response to the caller.

Require Encryption

Requires that a request's XML element (which is represented by an XPath expression) be encrypted. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services.

Prerequisites

1. Configure Integration Server: Set up keystores and truststores in Integration Server, as described in *Securing Communications with the Server* in the document *Administering webMethods Integration Server*.
2. Configure Mediator: In the Integration Server Administrator, navigate to Solutions > Mediator > Administration > General and complete the IS Keystore Name, IS Truststore Name and Alias (signing) fields, as described in *Keystore Configuration* in the document *Administering WebMethods Mediator*.

When this policy action is set for the virtual service, Mediator provides decryption of incoming requests and encryption of outgoing responses. Mediator can encrypt and decrypt only individual elements in the SOAP message body that are defined by the XPath expressions configured for the policy action. Mediator requires that requests contain the encrypted elements that match those in the XPath expression. You must encrypt the entire element, not just the data between the element tags. Mediator rejects requests if the element name is not encrypted.

⚠ **Important:** Do not encrypt the entire SOAP body because a SOAP request without an element will appear to Mediator to be malformed.

Mediator attempts to encrypt the response elements that match the XPath expressions with those defined for the policy. If the response does not have any elements that match the XPath expression, Mediator will not encrypt the response before sending. If the XPath expression resolves a portion of the response message, but Mediator cannot locate a certificate to encrypt the response, then Mediator sends a SOAP fault exception to the consumer and a Policy Violation event notification to CentraSite.

How Mediator Encrypts Responses

The Require Encryption action encrypts the response back to the client by dynamically setting a public key alias at run time. Mediator determines the public key alias as follows:

1. If Mediator can access the X.509 certificate of the client (based on the incoming request signature), it will use "useReqSigCert" as the public key alias.

   OR

2. If the Identify Consumer action is present in the policy (and it successfully identifies a consumer application), then Mediator will look for a public key alias with that consumer name in the "IS Keystore Name" property. The "IS Keystore Name" property is specified in the Integration Server Administrator, under Solutions > Mediator > Administration > General. This property should be set to an Integration Server keystore that Mediator will use.

   For an Identify Consumer action that allows for anonymous usage, Mediator does *not* require a consumer name in order to send encrypted responses. In this case, Mediator can use one of the following to encrypt the response in the following order, depending on what is present in the security element:

   - A signing certificate.
   - Consumer name.
   - WSS username, SAML token or X.509 certificate.
   - HTTP authorized user.

   OR

3. If Mediator can determine the current IS user from the request (i.e., if an Integration Server WS-Stack determined that Subject is present), then the first principal in that subject is used.

   OR

4. If the above steps all fail, then Mediator will use either the WS-Security username token or the HTTP Basic-Auth user name value. There should be a public key entry with the same name as the identified username.

Note: You can include this action multiple times in a single policy.

Input Parameters

`Namespace`	*String* Optional. Namespace of the element required to be encrypted. **Note:** Enter the namespace prefix in the following format: `xmlns:`*`<prefix-name>`*. For example: `xmlns:soapenv`. For more information, see the XML Namespaces specifications at **http://www.w3.org/TR/REC-xml-names/#ns-decl**. The generated XPath element in the policy should look similar to this:

```
<sp:SignedElements ↩
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
              <sp:XPath ↩
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">//soapenv:Body</sp:XPath
            </sp:SignedElements>
```

`Element Required to be Encrypted`	*String* An XPath expression that represents the XML element that is required to be encrypted.

Require HTTP Basic Authentication

This action uses HTTP Basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header. Mediator authorizes the credentials against the list of users registered in the Integration Server on which Mediator is running. This type of consumer authentication is referred to as "preemptive authentication". If you want to perform "preemptive authentication", a policy that includes this action must also include the Identify Consumer action.

If the user/password value in the Authorization header cannot be authenticated as a valid Integration Server user (or if the Authorization header is not present in the request), a 500 SOAP fault is returned, and the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. This type of consumer authentication is referred to as "non-preemptive authentication". If the client does not successfully respond to the challenge, a 401 "WWW-Authenticate: Basic" response is returned and the invocation is not routed to the policy engine. As a result, no events are recorded for that invocation, and its key performance indicator (KPI) data are not included in the performance metrics.

If you choose to omit the "Require HTTP Basic Authentication" action (and regardless of whether an Authorization header is present in the request or not), then:

- Mediator forwards the request to the native service, without attempting to authenticate the request.
- The native service returns a 401 "WWW-Authenticate: Basic" response, which Mediator will forward to the client; the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated.

In the case where a consumer sends a request with transport credentials (HTTP Basic authentication) and message credentials (WSS Username or WSS X.509 token), the message credentials take precedence over the transport credentials when Integration Server determines which credentials it should use for the session. For more information, see **Require WSS Username Token** and **Require WSS X.509 Token**. In addition, you must ensure that the service consumer that connects to the virtual service has an Integration Server user account.

Note: Do not include the “Require HTTP Basic Authentication” action in a virtual service's run-time policy if you selected the **OAuth2** option in the virtual service's Routing Protocol step.

Input Parameters

Note: This input parameter is not available in Mediator versions prior to 9.0.

`Authenticate Credentials`	Required. Authorizes consumers against the list of users registered in the Integration Server on which Mediator is running.

Require Signing

This action requires that a request's XML element (which is represented by an XPath expression) be signed. This action supports WS-SecurityPolicy 1.2.

Prerequisites

1. Configure Integration Server: Set up keystores and truststores in Integration Server, as described in *Securing Communications with the Server* in the document *Administering webMethods Integration Server*.
2. Configure Mediator: In the Integration Server Administrator, navigate to Solutions > Mediator > Administration > General and complete the IS Keystore Name, IS Truststore Name and Alias (signing) fields, as described in *Keystore Configuration* in the document *Administering WebMethods Mediator*. Mediator uses the signing alias specified in the Alias (signing) field to sign the response.

When this action is set for the virtual service, Mediator validates that the requests are properly signed, and provides signing for responses. Mediator provides support both for signing an entire SOAP message body or individual elements of the SOAP message body.

Mediator uses a digital signature element in the security header to verify that all elements matching the XPath expression were signed. If the request contains elements that were not signed or no signature is present, then Mediator rejects the request.

Notes:

1. You must map the public certificate of the key used to sign the request to an Integration Server user. If the certificate is not mapped, Mediator returns a SOAP fault to the caller.
2. You can include this action multiple times in a policy.

Input Parameters

`Namespace`	*String* Optional. Namespace of the element required to be signed. **Note:** Enter the namespace prefix in the following format: `xmlns:`*`<prefix-name>`*. For example: `xmlns:soapenv`. For more information, see the XML Namespaces specifications at **http://www.w3.org/TR/REC-xml-names/#ns-decl**. The generated XPath element in the policy should look similar to this:

```
<sp:SignedElements ↩
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
              <sp:XPath ↩
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">//soapenv:Body</sp:XPat
            </sp:SignedElements>
```

`Element Required to be Signed`	*String* An XPath expression that represents the XML element that is required to be signed.

Require SSL

Requires that requests be sent via SSL client certificates. This action supports WSSecurityPolicy 1.2 but can be used for both SOAP and REST services.

When this policy action is set for the virtual service, Mediator ensures that requests are sent to the server using the HTTPS protocol (SSL). The policy also specifies whether the client certificate is required. This allows Mediator to verify the client sending the request. If the policy requires the client certificate, but it is not presented, Mediator rejects the message.

When a client certificate is required by the policy, the Integration Server HTTPS port should be configured to request or require a client certificate.

Input Parameters

`Client Certificate Required`	*Boolean* Specifies whether client certificates are required for the purposes of: ■ Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests ■ Signing SOAP responses or encrypting SOAP responses

Value	**Description**

	`Yes`	Require client certificates.
	`No`	Default. Do not require client certificates.

Require Timestamps

Note: Dependency requirement: A policy that includes this action must also include *all* of the following actions: **Require SSL**, **Require Signing**, **Require Encryption**.

When this policy action is set for the virtual service, Mediator requires that timestamps be included in the request header. Mediator checks the timestamp value against the current time to ensure that the request is not an old message. This serves to protect your system against attempts at message tampering, such as replay attacks. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services.

Mediator rejects the request if either of the following happens:

- Mediator receives a timestamp that exceeds the time defined by the timestamp element.
- A timestamp element is not included in the request.

Input Parameters

None.

Require WSS SAML Token

When this action is set for a virtual service, Mediator uses a WSS Security Assertion Markup Language (SAML) assertion token to validate service consumers. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services and cannot be used with REST services.

For more information about configuring your system for SAML token processing, see *SAML Support in Mediator* in the document *Administering webMethods Mediator*.

Input Parameters

`SAML Subject Confirmation`	*String* Select one of the following SAML subject confirmation methods:	
	Value	**Description**
		Default. Select this option if consumers use the SAML V1.1 or V2.0 Holder-of-Key Web Browser SSO Profile, which allows for transport of holder-of-key assertions. In this scenario, the consumer presents a holder-of-key SAML assertion acquired from its preferred identity provider to access a web-based resource at a service provider. If you select `Holder of Key`, Mediator also implicitly selects the “timestamp” and “signing” assertions to the virtual service definition

	`Holder of Key`	(VSD). Thus, you should not add the ““Require Timestamps” and “Require Signing” policy actions to a virtual service if the “Require WSS SAML Token” action is already applied.
	`Bearer`	Select this option if consumers use SAML V1.1 Bearer token authentication, in which a Bearer token mechanism relies upon bearer semantics as a means by which the consumer conveys to Mediator the sender's identity. If you select `Bearer`, the “timestamp” and “signing” assertions will be added to the virtual service definition (VSD). **Note:** If consumers use SAML 2.0 Sender-Vouches tokens, configure your system as described in *SAML Support in Mediator* in the document *Administering WebMethods Mediator*.
`SAML Version`	*String* Specifies the WSS SAML Token version to use: 1.1 or 2.0.	

Require WSS Username Token

Note: Dependency requirement: A policy that includes this action must also include the Identify Consumer action.

When this policy action is set for the virtual service, Mediator uses WS-SecurityPolicy authentication to validate user names and passwords that are transmitted in the SOAP message header for the WSS Username token. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST services.

In the case where a consumer is sending a request with both transport credentials (HTTP basic authentication) and message credentials (WSS Username or X.509 token), the message credentials take precedent over the transport credentials when Integration Server is determining which credentials it should use for the session. For more information, see **Require HTTP Basic Authentication**.

Mediator rejects requests that do not include the username token and password of an Integration Server user. Mediator only supports clear text passwords with this kind of authentication

Input Parameters

None.

Require WSS X.509 Token

Note: Dependency requirement: A policy that includes this action must also include the Identify Consumer action.

Identifies consumers based on a WSS X.509 token. This action supports WSSecurityPolicy 1.2 and cannot be used with REST services.

In the case where a consumer is sending a request with both transport credentials (HTTP Basic authentication) and message credentials (WSS X.509 token or WSS Username), the message credentials take precedence over the transport credentials when Integration Server is determining which credentials it should use for the session. For more information, see **Require HTTP Basic Authentication**. In addition, you must ensure that the service consumer that connects to the virtual service has an Integration Server user account.

Input Parameters

None.

Throttling Traffic Optimization

Notes:

1. This action is not available in Mediator versions below 9.0.
2. Dependency requirement: A policy that includes this action must also include the **Identify Consumer** action if the `Limit Traffic for Applications` option is selected.

This action limits the number of service invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated.

Reasons for limiting the service invocation traffic include:

- To avoid overloading the back-end services and their infrastructure.
- To limit specific consumers in terms of resource usage (that is, you can use the “Monitor Service Level Agreement” action to monitor performance conditions for a particular consumer, together with “Throttling Traffic Optimization” to limit the resource usage).
- To shield vulnerable servers, services, and even specific operations.
- For service consumption metering (billable pay-per-use services).

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see the section *Configuring Communication with CentraSite* in the document *Administering webMethods Mediator*.

Input Parameters

`Soft Limit`	*Number* Optional. Specifies the maximum number of invocations allowed per `Interval` before issuing an alert. Reaching the soft limit will not affect further processing of requests (until the `Hard Limit` is reached). **Note:** The limit is reached when the total number of invocations coming from all *all* the consumer applications (specified in the `Limit Traffic for Applications` field) reaches the limit. Soft Limit is computed in an asynchronous manner; thus when multiple requests are made at the same time, it may be possible that the Soft Limit alert will not be strictly accurate.	
`Hard Limit`	*Number* Required. Specifies the maximum number of invocations allowed per alert interval before stopping the processing of further requests and issuing an alert. Typically, this number should be higher than the soft limit. **Note:** The limit is reached when the total number of invocations coming from all *all* the consumer applications (specified in the `Limit Traffic for Applications` field) reaches the limit. Hard Limit is computed in an asynchronous manner; thus when multiple requests are made at the same time, it may be possible that the Hard Limit alert will not be strictly accurate.	
`Limit Traffic for Applications`	*String* Specifies the consumer application(s) that this action applies to. To specify multiple consumer applications, use the + button to add rows, or select **Any Consumer** to apply this action to any consumer application.	
`Interval`	*Number* Specifies the amount of time for the soft limit and hard limit to be reached.	
`Frequency`	*String* Specifies how frequently to issue alerts.	
	Value	**Description**
	`Every Time`	Issue an alert every time the specified condition is violated.
	`Only Once`	Issue an alert only the first time the specified condition is violated.
`Reply To Destination`	*String* Optional. Specifies where to log the alerts. **Important:** Ensure that Mediator is configured to send event notifications to the destination(s) you specify here. For details, see *Alerts and Transaction Logging* in the document *Administering webMethods Mediator*.	
	Value	**Description**
		Sends the alerts to the virtual service's Events profile in CentraSite. Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see the section *Configuring Communication*

	`CentraSite`	*with CentraSite* in the document *Administering webMethods Mediator*.
	`Local Log`	Sends the alerts to the server log of the Integration Server on which Mediator is running. Also choose a value in the `Log Level` field: ■ Info: Logs error-level, warning-level, and informational-level alerts. ■ Warn: Logs error-level and warning-level alerts. ■ Error: Logs only error-level alerts. **Important:** The Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to **Settings > Logging > Server Logger**).
	`SNMP`	Sends the alerts to CentraSite's SNMP server or a third-party SNMP server. Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > Email**). For the procedure, see the section *SNMP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.
	`Email`	Sends the alerts to an SMTP email server, which sends them to the email address(es) you specify here. To specify multiple addresses, use the ⊞ button to add rows. Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > Email**). For the procedure, see the section *SMTP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.
`Alert Message for Soft Limit`	*String* Optional. Specify a text message to include in the soft limit alert.	
`Alert Message for Hard Limit`	*String* Optional. Specify a text message to include in the hard limit alert.	

Validate Schema

This action validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Mediator can enforce this policy action for messages sent between services. When this policy is set for the virtual service, Mediator validates XML request messages, response messages, or both, against the XML schema referenced in the WSDL.

Input Parameters

<table>
<tr><td rowspan="4"><code>Validate SOAP Message(s)</code></td><td colspan="2"><em>Object</em> Validates request and/or response messages. You may select both Request and Response.</td></tr>
<tr><td><strong>Value</strong></td><td><strong>Description</strong></td></tr>
<tr><td><code>Request</code></td><td>Validate all requests.</td></tr>
<tr><td><code>Response</code></td><td>Validate all responses.</td></tr>
</table>

Important: Be aware that Mediator does not remove `wsu:Id` attributes that may have been added to a request by a consumer as a result of security operations against request elements (i.e., signatures and encryptions). In this case, to avoid schema validation failures you would have to add a Request Processing step to the virtual service so that the requests are passed to an XSLT transformation file that removes the `wsu:Id` attribute. For details about the Request Processing step, see the section *Virtual Services in CentraSite Control* .

3 Built-In Run-Time Actions Reference for Virtual APIs

This section describes the built-in run-time actions that you can include in run-time governance rules for virtualized APIs. You use these actions only when you are using the CentraSite Business UI to create run-time policies for virtualized APIs. The content is organized under the following sections:

Summary of the Run-Time Actions for Virtual APIs

You can include the following kinds of built-in run-time actions in the run-time governance rules for virtualized APIs:

- Request Handling Actions
- Policy Enforcement Actions
- Response Handling Actions
- Error Handling Action

Request Handling Actions

Mediator provides the following actions for handling requests:

Request HTTP Protocol	Specifies the protocol (HTTP or HTTPS) for the virtualized API to accept requests. In addition, *For SOAP APIs.* Specify the SOAP version. *For REST APIs.* Specify the HTTP method.
Request Transformation	Invokes an XSLT transformation in the SOAP request before it is submitted to the native API.
Invoke webMethods IS Service	Invokes a webMethods IS service to preprocess the request before it is submitted to the native API.

Policy Enforcement Actions

Mediator provides the following categories of policy enforcement actions:

- Logging and Monitoring Actions
- Routing Actions
- Security Actions

- Validation Action

Logging and Monitoring Actions

Log Invocations	Logs request/response payloads to a destination you specify.
Monitor Service Level Agreement	Monitors the run-time performance of a virtual alias, especially for particular consumer(s). You can configure this action to define a Service Level Agreement (SLA), which is set of conditions that define the level of performance that a specified consumer should expect from the alias.

Routing Actions

Endpoint Properties	Defines a set of properties for an endpoint to which you route requests. You can specify a SOAP optimization method, timeouts for HTTP connections and socket reads, the SSL client authentication aliases for the endpoint (Client Certificate Alias, Keystore Alias, Truststore Alias), and the WS-Security headers of the requests that Mediator should pass to the native API.
Set Headers	Specifies the HTTP headers to authenticate the requests.
Set HTTP Authorization	Specifies the authentication scheme (HTTP Basic authentication, NTLM or OAuth).
Straight Through Routing	Routes the requests directly to a native endpoint that you specify.

Security Actions

Allow Anonymous Usage	Allows anonymous users to access the APIs.
Evaluate Client Certificate for SSL Connectivity	Mediator will validate the client's certificate that the consumer application submits to the API in CentraSite. The client certificate that is used to identify the consumer is supplied by the client to the Mediator during the SSL handshake over the transport layer.
Evaluate Hostname	Mediator will try to identify the consumer's hostname against either the Registered Consumers list (the list of consumers available in Mediator) or the Global Consumers list (the list of Registered Consumers).
Evaluate HTTP Basic Authentication	You can select one of the following options: ■ Mediator will try to identify the consumer against either the Registered Consumers list (the list of consumers available in Mediator) or the Global Consumers list (the list of Registered Consumers). ■ Mediator will try to verify the consumer's authentication credentials contained in the request's Authorization header against the list of users registered in the Integration Server on which Mediator is running.
Evaluate IP Address	Mediator will try to identify the consumer's IP address against either the Registered Consumers list (the list of consumers available in Mediator) or the Global Consumers list (the list of Registered Consumers).

Evaluate WSS Username Token	*For SOAP APIs.* Mediator will try to identify the consumer's WSS username token against either the Registered Consumers list (the list of consumers available in Mediator) or the Global Consumers list (the list of Registered Consumers).
Evaluate X.509 Certificate	*For SOAP APIs.* Mediator will try to identify the consumer's WSS X.509 token against either the Registered Consumers list (the list of consumers available in Mediator) or the Global Consumers list (the list of Registered Consumers).
Evaluate XPath Address	Mediator will try to identify the consumer's XPath expression against either the Registered Consumers list (the list of consumers available in Mediator) or the Global Consumers list (the list of Registered Consumers).
Require SSL	*For SOAP APIs.* Requires that requests be sent via SSL client certificates.

Validation Action

Validate Schema	Validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Response Handling Actions

Response Transformation	Invokes an XSLT transformation in the SOAP response payloads from XML format to the format required by the consumer.
Invoke webMethods IS Service	Invokes a webMethods IS service to process the response from the native API before it is returned to the consumer.

Error Handling Action

Custom SOAP Response Message	Returns a custom error message (and/or the native provider's service fault content) to the consumer when the native provider returns a service fault.

Action Evaluation Order and Dependencies for Virtual APIs

When you publish a virtual API, CentraSite automatically validates the API's policy enforcement workflow to ensure that:

- Any action that appears in a single message flow multiple times is allowed to appear multiple times.

  For those actions that can appear in a message flow only once (for example, Evaluate IP Address), Mediator will choose only one, which might cause problems or unintended results.

- All action dependencies are properly met. That is, some actions must be used in conjunction with another particular action.

CentraSite will inform you of any violation, and you will need to correct the violations before publishing the API.

Effective Policies

When you publish a virtual endpoint to Mediator, CentraSite combines the actions specified within the virtual endpoint's enforcement definition, and generates what is called the effective policy for the virtual endpoint. For example, suppose your virtual endpoint is configured with two run-time actions: one that performs a logging action and another that performs a security action. When you publish the virtual endpoint, CentraSite automatically combines the two actions into one effective policy. The effective policy, which contains both the logging action and the security action, is the policy that CentraSite actually publishes to Mediator with the virtual endpoint.

When CentraSite generates the effective policy, it validates the resulting action list to ensure that it contains no conflicting or incompatible actions. If the list contains conflicts or inconsistencies, CentraSite resolves them according to Policy Resolution Rules.

The effective policy that CentraSite produces for a virtual endpoint is contained in an object called a virtual service definition (VSD). The VSD is given to Mediator when you publish the virtual endpoint. After you publish a virtual endpoint, you can view its VSD (and thus examine the effective policy that CentraSite generated for it) from the Mediator user interface.

The following table shows:

- The order in which Mediator evaluates the actions.
- Action dependencies (that is, whether an action must be used in conjunction with another particular action).

Evaluation Order	Action	Dependency	Can include multiple times in a policy if the selection criteria is combined using an *AND* operator, not an *OR*?
1	Evaluate HTTP Basic Authentication	None.	No. Mediator includes only one action in the effective policy.
2	Evaluate WSS Username Token	None. If you select this action in addition to other actions, you must select the ALL option to join the identifiers with the AND operator.	No. Mediator includes only one action in the effective policy.
3	Evaluate X.509 Certificate	None. If you select this action in addition to other	No. Mediator includes only one action in the effective policy.

Evaluation Order	Action	Dependency	Can include multiple times in a policy if the selection criteria is combined using an *AND* operator, not an *OR*?
		actions, you must select the ALL option to join the identifiers with the AND operator.	
4	Evaluate IP Address	None.	No. Mediator includes only one action in the effective policy.
5	Evaluate XPath Address	None.	No. Mediator includes only one action in the effective policy.
6	Evaluate Hostname	None.	No. Mediator includes only one action in the effective policy.
7	Require SSL	None.	If multiple actions appear, and one of them has its Client Certificate Required parameter set to Yes, only one occurrence of the action appears in the effective policy.
8	Validate Schema	None.	If at least one occurrence of the action is configured to validate requests, and at least one occurrence of the action is configured to validate responses, then Mediator includes in the effective policy an action to validate both requests and responses. Otherwise, an action is chosen which validates only requests or only responses (depending on the value of the Validate SOAP Messages parameter of the action).
9	Log Invocations	None.	No. Mediator includes only one action in the effective policy.
10	Monitor Service Level Agreement	At least one of the Evaluate actions.	Yes. Mediator includes all Monitor Service Level Agreement actions in the effective policy.

Run-Time Actions Reference for Virtual APIs

This section provides an alphabetic list of the built-in run-time actions you can include in run-time governance rules for virtualized APIs:

- Allow Anonymous Usage
- Custom SOAP Response Message
- Endpoint Properties
- Evaluate Client Certificate for SSL Connectivity
- Evaluate Hostname

- Evaluate HTTP Basic Authentication
- Evaluate IP Address
- Evaluate WSS Username Token
- Evaluate WSS X.509 Certificate
- Evaluate XPath Address
- Invoke webMethods IS Service
- Log Invocations
- Monitor Service Level Agreement
- Response Transformation
- Request HTTP Protocol
- Request Transformation
- Require SSL
- Set Headers
- Set HTTP Authentication
- Straight Through Routing
- Validate Schema

Allow Anonymous Usage

This action allows anonymous users to access the APIs.

Input Parameters

`Allow Anonymous Usage`	*Boolean*. Specifies whether to allow all users to access the API, without restriction.	
	Value	**Description**
	`True`	Default. Allows only the identified users to access the API.
	`False`	Allow all users to access the API.

Custom SOAP Response Message

This action returns a custom error response (and/or the native provider's service fault content) to the consumer when the native provider returns a service fault. Alternatively, you can configure global error responses for all virtual services, using Mediator's Service Fault Configuration page (see *Configuring Global Service Fault Responses* in the document *Administering webMethods Mediator*).

Input Parameters

<table>
<tr><td><code>Failure Message</code></td><td><i>String.</i> Returns the fault responses to the consumer, when:
■ When a fault is returned by the native API provider.
In this case, the $ERROR_MESSAGE variable in the fault response will contain the message produced by the provider's exception that caused the error. This is equivalent to the getMessage call on the Java Exception. This maps to the faultString element for SOAP 1.1 or the Reason element for SOAP 1.2 catch. Mediator discards the native API provider's fault and does not return this content to the web service caller since it could be considered a security issue, especially if the native provider is returning a stack trace with its response.
■ When a fault is returned by internal Mediator exceptions (such as policy violation errors, timeouts, etc.).
In this case, $ERROR_MESSAGE will contain the error message generated by Mediator.</td></tr>
<tr><td><code>Send Native Failure Message</code></td><td><i>Boolean. Optional.</i> Specifies whether to send native SOAP / REST failure message to the consumer.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>True</code></td><td>Default. Mediator sends the failure message to the consumer.</td></tr>
<tr><td><code>False</code></td><td>Mediator does not send the failure message to the consumer.</td></tr>
</table></td></tr>
<tr><td><code>Pre-processing webMethods IS Service</code></td><td><i>String. Optional.</i> Invokes one or more webMethods IS services to manipulate the response message before the Custom SOAP Response Message action is invoked. The IS service will have access to the response message context (the axis2 MessageContext instance) before it is updated with the custom error message. For example, you might want to send emails or perform custom alerts based on the response payload.</td></tr>
<tr><td><code>Post-processing webMethods IS Service</code></td><td><i>String.</i> Optional. Invokes one or more webMethods IS services to manipulate the API fault after the Custom SOAP Response Message action is invoked. The IS service will have access to the entire API fault and the custom error message. You can make further changes to the fault message structure, if needed.</td></tr>
</table>

Endpoint Properties

This action defines a set of properties for an endpoint to which you route requests.

Input Parameters

`SOAP Optimization Method`	*String. Optional.* For a SOAP endpoint. Specifies the optimization methods to optimize the payloads of SOAP requests:	
	Value	**Description**
	`MTOM`	Default. Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment, and will forward the attachment to the native API.
	`SWA`	Indicates that Mediator expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native API.
`HTTP Connection Timeout`	*String.* The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file *Integration Server_directory\packages\WmMediator\config\resources\pg-config.properties* . The default of that property is 30 seconds.	
`Read Timeout`	*Number Optional.* The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *Integration Server_directory\packages\WmMediator\config\resources\pg-config.properties* . The default of that property is 30 seconds.	
`SSL Configuration`	*Object.* Enables SSL client authentication for the endpoint.	
	Value	**Description**
	`Client Certificate Alias`	The client's private key to be used for performing SSL client authentication.
	`Truststore Alias`	The truststore alias of the instance of Integration Server on which Mediator is running.
	`Keystore Alias`	The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of Client Certificate Alias) will be used for performing SSL client authentication.
`WSS Header`	*String.* Specifies WS-Security headers of the incoming requests that Mediator should pass to the native API.	

Evaluate Client Certificate for SSL Connectivity

When this action is configured for a virtual API, Mediator validates the client's certificate that the consumer application submits to the API in CentraSite. The client certificate that is used to identify the consumer is supplied by the client to the Mediator during the SSL handshake over the transport layer. In order to identify consumers by transport-level certificates, the run-time communication between the client and the Mediator must be over HTTPS and the client must pass a valid certificate.

To use this action, the following prerequisites must be met:

- In Integration Server, create an HTTPS port, as described in *Configuring Ports* in the *webMethods Integration Server Administrator's Guide*.
- In Integration Server, create a keystore and truststore, as described in *Securing Communications with the Server* in the *webMethods Integration Server Administrator's Guide*.
- Configure Mediator by setting the IS Keystore and IS Truststore parameters, as described in *Configuring Mediator > Keystore Configuration* in the document *Administering webMethods Mediator*.
- Configure Mediator by setting the HTTPS Ports Configuration parameter, as described in *Configuring Mediator > Ports Configuration* in the document *Administering webMethods Mediator*.

Mediator rejects requests that do not include a client certificate during the SSL handshake over the Transport layer.

Input Parameters

`Identify Consumer`	*String.* The list of consumers against which the client certificate should be validated for identifying requests from a particular consumer.	
	Value	**Description**
	`Registered Consumers`	Mediator will try to verify the consumer's certificate against the list of consumer applications who are registered as consumers for the API.
	`Global Consumers`	*Default.* Mediator will try to verify the consumer's certificate against a list of users registered in the Integration Server on which Mediator is running.

If Mediator cannot identify the consumer, Mediator fails the request and generates a Policy Violation event.

Evaluate Hostname

If you select this action, Mediator will evaluate the request to ensure that the request originated from the particular host machine. Mediator identifies the consumer against the list of users registered in the Integration Server on which Mediator is running.

Input Parameters

`Identify User Using HostName Address`	*String. Optional.*	
	Value	**Description**
	`Registered Consumers`	Mediator will try to identify the consumer's hostname against the list of Registered Consumers.
	`Global Consumers`	Mediator will try to identify the consumer's hostname against the consumers available in Mediator.

If Mediator cannot identify the consumer, Mediator fails the request and generates a Policy Violation event.

Evaluate HTTP Basic Authentication

If you set `Validate User Using HTTP Basic Authentication` to True, this type of consumer authentication is referred to as “preemptive authentication”.

If the user/password value in the Authorization header cannot be authenticated as a valid Integration Server user (or if the Authorization header is not present in the request), a 500 SOAP fault is returned, and the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. This type of consumer authentication is referred to as “non-preemptive authentication”. If the client does not successfully respond to the challenge, a 401 “WWW-Authenticate: Basic” response is returned and the invocation is not routed to the policy engine. As a result, no events are recorded for that invocation, and its key performance indicator (KPI) data are not included in the performance metrics.

Input Parameters

`Identify User Using HTTP Basic Authentication`	*String. Optional.*	
	Value	**Description**
	`Registered ↩ Consumers`	Mediator will try to identify the consumer against the list of Registered Consumers.

	`Global Consumers`	Mediator will try to identify the consumer against the consumers available in Mediator.
`Validate User Using HTTP Basic Authentication`	*Boolean. Optional.*	
	Value	**Description**
	`True`	Default. Mediator will verify the consumer's authentication credentials contained in the request's Authorization header against the list of users registered in the Integration Server on which Mediator is running.
	`False`	Mediator will not verify the consumer's authentication credentials.

If Mediator cannot validate or identify the consumer, Mediator fails the request and generates a Policy Violation event.

Evaluate IP Address

If you select this action, Mediator will evaluate the request to ensure that the request header contains the X-Forwarded-For, which is used for identifying the IP address of a consumer through an HTTP proxy.

Input Parameters

`Identify User Using IP Address`	*String. Optional.*	
	Value	**Description**
	`Registered Consumers`	Mediator will try to identify the consumer's IP address against the list of Registered Consumers. Mediator will evaluate whether the request header contains the X-Forwarded-For, which is used for identifying the IP address of a consumer through an HTTP proxy.
	`Global Consumers`	Mediator will try to identify the consumer's IP address against the consumers available in Mediator.

Evaluate WSS Username Token

If you select this action, Mediator will evaluate the request to ensure that the request header contains the WSS username token, which is used for identifying a consumer.

Input Parameters

<table>
<tr><td rowspan="4"><code>Identify User Using WSS Username Token</code></td><td colspan="2"><i>String. Optional.</i></td></tr>
<tr><td><b>Value</b></td><td><b>Description</b></td></tr>
<tr><td><code>Registered Consumers</code></td><td>Mediator will try to identify the consumer's WSS username token against the list of Registered Consumers.</td></tr>
<tr><td><code>Global Consumers</code></td><td>Mediator will try to identify the consumer's WSS username token against the consumers available in Mediator.</td></tr>
</table>

If Mediator cannot identify the consumer, Mediator fails the request and generates a Policy Violation event.

Evaluate WSS X.509 Certificate

If you select this action, Mediator will evaluate the request to ensure that the request header contains the WSS X.509 token, which is used for identifying a consumer.

Input Parameters

<table>
<tr><td rowspan="4"><code>Identify User Using WSS X.509 Token</code></td><td colspan="2"><i>String. Optional.</i></td></tr>
<tr><td><b>Value</b></td><td><b>Description</b></td></tr>
<tr><td><code>Registered Consumers</code></td><td>Mediator will try to identify the consumer's WSS X.509 token against the list of Registered Consumers.</td></tr>
<tr><td><code>Global Consumers</code></td><td>Mediator will try to identify the consumer's WSS X.509 token against the consumers available in Mediator.</td></tr>
</table>

If Mediator cannot identify the consumer, Mediator fails the request and generates a Policy Violation event.

Evaluate XPath Address

If you select this action, Mediator will evaluate the request to ensure that the request header contains an XPath expression, and that expression matches with the expression defined in the consumer details.

Input Parameters

`Identify User Using XPath Address`	*String. Optional.*	
	Value	**Description**
	`Registered Consumers`	Mediator will try to identify the consumer's XPath expression against the list of Registered Consumers.
	`Global Consumers`	Mediator will try to identify the consumer's XPath expression against the consumers available in Mediator.
`Namespace`	The namespace of the XPath expression.	
`XPath Expression`	An argument for evaluating the XPath expression.	

If Mediator cannot identify the consumer, Mediator fails the request and generates a Policy Violation event.

Invoke webMethods IS Service

This action invokes a webMethods IS service to preprocess the request before it is submitted to the native API.

Input Parameters

`IS Service`	*String.* Specifies the webMethods IS service.

Log Invocations

This action logs request/response payloads. You can specify the log destination and the logging frequency. This action also logs other information about the requests/responses, such as the API name, operation name, the Integration Server user, a timestamp, and the response time.

Input Parameters

`Request Payloads`	*Boolean. Optional.* Specifies whether to log all request payloads.	
	Value	**Description**
	`True`	Log all request payloads.
	`False`	Do not log request payloads.
`Response Payloads`	*Boolean. Optional.* Specifies whether to log all response payloads.	
	Value	**Description**

	`True`	Log all response payloads.
	`False`	Do not log response payloads.
`Log Generation Frequency`	*String*. Specifies how frequently to log the payload.	
	Value	**Description**
	`None`	Default. Do not log payloads.
	`Always`	Log all requests and/or responses.
	`On Success`	Log only the successful responses and/or requests.
	`On Failure`	Log only the failed requests and/or responses.
`Send Data To`	*String*. Specifies where to log the payload. **Important:** Ensure that Mediator is configured to log the payloads to the destination(s) you specify here. For details, see *Alerts and Transaction Logging* in the document *Administering webMethods Mediator*.	
	Value	**Description**
	`CentraSite`	Logs the payloads in the API's Events profile in CentraSite. Prerequisite: You must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see the section *Configuring Communication with CentraSite* in the document *Administering webMethods Mediator*.
	`Local Log`	Logs the payloads in the server log of the Integration Server on which Mediator is running. Also choose a value in the `Log Level` field: ■ Info: Logs error-level, warning-level, and informational-level alerts. ■ Warn: Logs error-level and warning-level alerts. ■ Error: Logs only error-level alerts. **Important:** The Integration Server Administrator's logging level for Mediator should match the logging level specified for this action (go to **Settings > Logging > Server Logger**).

	`SNMP`	Logs the payloads in CentraSite's SNMP server or a third-party SNMP server. Prerequisite: You must configure the SNMP server destination (in the Integration Server Administrator, go to **Solutions** > **Mediator** > **Administration** > **SNMP**). For the procedure, see the section *SNMP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.
	`Email`	Sends the payloads to an SMTP email server, which sends them to the email address(es) you specify here. Mediator sends the payloads as email attachments that are compressed using gzip data compression. To specify multiple addresses, use the + button to add rows. Prerequisite: You must configure the SMTP server destination (in the Integration Server Administrator, go to **Solutions** > **Mediator** > **Administration** > **Email**). For the procedure, see the section *SMTP Destinations for Run-Time Events* in the document *Administering webMethods Mediator*.
	`Audit Log`	Logs the payload to the Integration Server audit logger. For information, see the *webMethods Audit Logging Guide*. **Note:** If you expect a high volume of events in your system, it is recommended that you select the Audit Log destination for this action.

Monitor Service Level Agreement

Monitors the run-time performance conditions for a virtual API for one or more specified consumers , and then send alerts when the performance conditions are violated

You can configure this action to define a *Service Level Agreement (SLA)*, which is a set of conditions that defines the level of performance that a consumer should expect from a service. You can use this action to identify whether a service's threshold rules are met or exceeded. For example, you might define an agreement with a particular consumer that sends an alert to the consumer if responses are not sent within a certain maximum response time. You can configure SLAs for each virtual service/consumer application combination.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), Mediator sends an alert as soon as the performance condition is violated, without having to wait until the end of the metrics tracking interval. You can choose whether to send an alert only once during the interval, or every time the violation occurs during the interval. (Mediator will send another alert the next time a condition is violated during a subsequent interval.) For information about the the metrics tracking interval, see *The Metrics Tracking Interval* .

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), Mediator aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

This action does not include metrics for failed invocations.

Note: To enable Mediator to publish performance metrics, you must configure Mediator to communicate with CentraSite (in the Integration Server Administrator, go to **Solutions > Mediator > Administration > CentraSite Communication**). For the procedure, see the section *Configuring Communication with CentraSite* in the document *Administering webMethods Mediator*.

Input Parameters

<table>
<tr><td><code>Action Configuration</code></td><td colspan="2"><i>Object.</i> Specifies one or more conditions to monitor. To do this, specify a metric, operator, and value for each metric. To specify multiple conditions, use the ➕ button to add multiple rows. If multiple parameters are used, they are connected by the AND operator.</td></tr>
<tr><td rowspan="5"><code>Name</code></td><td colspan="2"><i>String Array.</i> The metrics to monitor.</td></tr>
<tr><td><b>Value</b></td><td><b>Description</b></td></tr>
<tr><td><code>None</code></td><td>Default.</td></tr>
<tr><td><code>Availability</code></td><td>Indicates whether the service was available to the specified consumers in the current interval.</td></tr>
<tr><td><code>Average Response Time</code></td><td>The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment Mediator receives the request until the moment it returns the response to the caller.</td></tr>
</table>

Response Transformation

This action invokes an XSLT transformation in the SOAP response payloads from XML format to the format required by the consumer.

Input Parameters

`Transformation File`	*Object.* Specifies the XSLT transformation file.

Request HTTP Protocol

This action specifies the protocol (HTTP or HTTPS), SOAP format, and the HTTP method for the virtual API to accept and process the requests.

Note: In order to have the API secured, it is mandatory that at least one of the (HTTP / HTTPS) protocol is set to “TRUE”.

Input Parameters

`Is SSL Enabled?`	*Boolean.* Specifies whether the virtual API is secured by HTTPS (Secure Sockets Layer (SSL)) authentication.	
	Value	**Description**
	`True`	The virtual API is secured by HTTPS (Secure Sockets Layer (SSL)).
	`False`	Default. The virtual API is not secured by HTTPS.
`SOAP Version`	*String. For SOAP APIs.* Specifies the SOAP format (None, SOAP 1.1, SOAP 1.2) of the requests that the virtual API will accept. Default: SOAP 1.1.	
`HTTP Method`	*String. Optional. For REST APIs.* Specifies the HTTP methods (GET, POST, PUT, DELETE) that the virtual API should be allowed to perform on a REST resource. Default: POST.	
`HTTP Protocol`	*Boolean.* Specifies whether the virtual API is secured by HTTP authentication.	
	Value	**Description**
	`True`	The virtual API is secured by HTTP.
	`False`	Default. The virtual API is not secured by HTTP.

Request Transformation

This action invokes an XSLT transformation in the SOAP request before it is submitted to the native API.

Input Parameters

`Transformation File`	*Object.* Specifies the XSLT transformation file.

Require SSL

Requires that requests be sent via SSL client certificates. This action supports WSSecurityPolicy 1.2 but can be used for both SOAP and REST services.

When this policy action is set for the virtual service, Mediator ensures that requests are sent to the server using the HTTPS protocol (SSL). The policy also specifies whether the client certificate is required. This allows Mediator to verify the client sending the request. If the policy requires the client certificate, but it is not presented, Mediator rejects the message.

When a client certificate is required by the policy, the Integration Server HTTPS port should be configured to request or require a client certificate.

Input Parameters

`Client Certificate Required`	*Boolean.* Specifies whether client certificates are required for the purposes of: ■ Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests ■ Signing SOAP responses or encrypting SOAP responses	
	Value	**Description**
	`Yes`	Require client certificates.
	`No`	Default. Do not require client certificates.

Set Headers

This action specifies the HTTP headers to authenticate the incoming requests.

Input Parameters

`Set Headers`	*String.* Specifies the HTTP headers to authenticate the requests.	
	Value	**Description**
	`None`	Default. Do not use any HTTP headers to authenticate requests
	`Reuse Existing Headers`	Use the HTTP headers that are contained in the requests.
	`Custom Headers`	Use the HTTP headers that you specify in the Name and Value columns below. If you need to specify multiple headers, use the + button to add rows.
`Header`	*Object.* Specifies the custom HTTP header(s) and the values.	
	Value	**Description**

	`Name`	Name of the HTTP header.
	`Value`	A value for the HTTP header.

Set HTTP Authentication

This action specifies the authentication scheme for incoming requests.

Input Parameters

<table>
<tr><td rowspan="5"><code>HTTP Authentication</code></td><td colspan="2"><em>String.</em> Authenticates requests to the native endpoint.</td></tr>
<tr><td><strong>Value</strong></td><td><strong>Description</strong></td></tr>
<tr><td><code>Reuse Existing Credentials</code></td><td>Default. Authenticates requests based on the credentials specified in the HTTP header.</td></tr>
<tr><td><code>Is Anonymous</code></td><td>Do not authenticate requests to the native endpoint.</td></tr>
<tr><td><code>Custom Credentials</code></td><td>Authenticates requests based on the credentials you specify in the <strong>Username</strong>, <strong>Password</strong> and <strong>Domain</strong> fields.</td></tr>
<tr><td><code>Authentication Scheme</code></td><td colspan="2"><em>String. Optional.</em> Specifies the mode of authentication: None, Basic Authentication (default), or NTLM (Windows only).
If you are choosing <strong>None</strong>, select the following option:
■ <strong>Is Anonymous</strong>: Does not authenticate requests.
If you are choosing <strong>Basic Authentication</strong> (default), select the following options:
■ <strong>Reuse Existing Credentials</strong>: Default. Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the “Authorization” header present in the original client request to the native service.
■ <strong>Custom Credentials</strong>: Authenticates requests according to the values you specify in the User, Password and Domain fields.
If you are choosing <strong>NTLM</strong> (Currently Windows only), select the following options:
<strong>Note:</strong> Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as “Windows”, then “NTLM” should be in its list. The “Negotiate” handshake will be supported in the near future. This note applies to all three options for NTLM.
■ <strong>Reuse Existing Credentials</strong>: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server.
■ <strong>Custom Credentials</strong>: Mediator uses the values you specify in the <strong>User</strong>, <strong>Password</strong> and <strong>Domain</strong> fields for an NTLM handshake with the server.</td></tr>
</table>

`OAuth2 Token`	*String. Optional.* If you are choosing **OAuth2**, select the following options: ■ **Reuse Existing Credentials**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server. ■ **Custom Credentials**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. **Note:** 1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *`Integration Server_directory`*\config directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time action "Evaluate HTTP Basic Authentication" will not be enforced when using the authentication scheme **OAuth2**. Specifies an OAuth2 access token to be deployed by Mediator.

Straight Through Routing

This action routes the requests directly to a native endpoint that you specify.

Input Parameters

`Endpoint`	*String.* Specifies the URL of the native endpoint to route the request to. For example:
	`http://mycontainer/creditCheckService`
	Alternatively, Mediator offers "Local Optimization" capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the **Default To** field the Routing Protocols tab, specify the native service in either of the following forms:
	`local://<Service-full-path>`
	OR
	`local://<server>:<port>/ws/<Service-full-path>`
	For example:
	`local://MediatorTestServices:NewMediatorTestServices_Port`

	which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server.

Validate Schema

This action validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Mediator can enforce this policy action for messages sent between services. When this policy is set for the virtual service, Mediator validates XML request messages, response messages, or both, against the XML schema referenced in the WSDL.

Input Parameters

<table>
<tr><td rowspan="4"><code>Validate SOAP Message(s)</code></td><td colspan="2"><i>Object.</i> Validates request and/or response messages. You may select both Request and Response.</td></tr>
<tr><td><b>Value</b></td><td><b>Description</b></td></tr>
<tr><td><code>Request</code></td><td>Validate all requests.</td></tr>
<tr><td><code>Response</code></td><td>Validate all responses.</td></tr>
</table>

Important: Be aware that Mediator does not remove `wsu:Id` attributes that may have been added to a request by a consumer as a result of security operations against request elements (i.e., signatures and encryptions). In this case, to avoid schema validation failures you would have to add a Request Transformation action or a Response Transformation action to the virtual service so that the requests are passed to an XSLT transformation file that removes the `wsu:Id` attribute. For details about the Request Transformation and Response Transformation actions, see ***Request Transformation*** and ***Response Transformation***.