Communicating between Software AG Products Using Event Routing

Version 10.11

October 2021

WEBMETHODS

This document applies to webMethods Event Routing 10.11 and to all subsequent releases.

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

Document ID: EDA-AG-1011-20211015

Table of Contents

About this Guide

This document gives you an overview of webMethods Event Routing, which is Software AG's framework for managing simple event-based interactions and more complex event analysis for pattern matching in real-time.

Event Routing offers the following key features and functionality:

- It is a solution for creating, processing, and monitoring events.
- It provides the infrastructure to rapidly build and adapt event-driven applications.
- It improves an organization's ability to comprehend the current state of the physical world and business environment and react rapidly to changes.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at https://documentation.softwareag.com.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at https://empower.softwareag.com/.

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

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

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

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.aspx and give us a call.

Software AG Tech Community

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

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

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

Event Routing Deprecation

webMethods Event Routing is deprecated. Contact Software AG Global Support for information about a replacement product.

1 Understanding Event Routing

This section contains an overview of Event Routing's concepts and usage. It also provides a list of related Software AG products and components, which can be used for building an event-driven system.

What is Event-Driven Architecture?

Using the event-driven architecture pattern, you can create systems for processing the events that shape your everyday business environment. An *event* can be something as simple as an electrical component being switched on or off, or more complicated, such as a bid being made in an auction house for the painting of a great master. An event represents something that has happened, and it may or may not require some follow-up action to be taken. An event can also represent something that was expected to happen but has failed to happen.

We all experience an event driven world every day. We walk through an airport and hear announcements of planes arriving and departing (these are events). The announcements (events) are emitted even if nobody is listening. If, however, the announcement is for my plane, then I will accept it, and start taking an action. My action may be to run to the gate, while someone else's activity might be to walk and get a snack. This is the basis of event-driven architecture: events are emitted and listeners can either take action on them or ignore them. The action I take is totally self-contained and does not rely on the activity of another person.

The term *event-driven* indicates that when an event happens, it can have a significance which requires some follow-up action to be taken. An event can be noticed by several observers or listeners, and each observer can react to the event differently. For one observer, an event might represent some critical status which requires immediate action. For another observer the same event might not be relevant at all.

The significance of a single event is sometimes only visible when viewed in the context of other events that together form a pattern. For example, if cash is withdrawn at a cash machine in the city center, this is not unusual, but if cash is withdrawn at many different cash machines on the same day throughout the city using the same card, this might raise the suspicion that the card is stolen.

If we change the focus from everyday events that we observe in the world around us to events that can have an influence on the way a company does its business, we can see that events such as the following could trigger a component in a predefined workflow:

- A trade order has been issued.
- A reading of sensor data (e.g. GPS, temperature or RFID reader) has occurred.
- A business process has reached completion.
- A software component has started successfully.

The existence of an event can be the trigger for processes, such as the invocation of a service, the initiation of a business process, or the publication of relevant information. Event Routing picks up on these ideas and provides a set of concepts for dealing with events at all stages throughout the processing chain.

Related Software AG Products and Components

The generic event driven architecture approach can be implemented by using different components for various roles, such as a messaging bus to transport the events, a common repository to hold the event type definitions, and a dashboarding tool to display the events.

The following table lists the products and components that Software AG provides for these roles:

Role in the event-driven architecture	Software AG component	Additional information
Message broker	Software AG Universal Messaging	
Event framework	Event Routing	Event Routing is Software AG's solution for event routing in the Software AG Common Platform.
Service bus	webMethods Integration Server	Integration Server is Software AG's standards-based enterprise service bus that integrates technologies from any vendor, including ERP systems, databases, mainframes, Web services, and others. It provides built-in services for Event Routing.
Event type repository	Event Type Store (run-time component)	The Event Type Store is a run-time repository that contains schemas of the events on the messaging bus. The event types are required in order to interpret the payload of events on the messaging bus.
	CentraSite Registry Repository (design-time component)	CentraSite enables you to archive, categorize and govern event type definitions.
Event type development tool	Software AG Designer, Events Development perspective	An Eclipse-based tool for creating and maintaining event types.
CEP Engine	Apama Correlator	Apama's event correlator is the engine that executes the sophisticated event pattern-matching logic that you

Role in the event-driven architecture	Software AG component	Additional information
		define in your Apama application.
CEP application design tool	Apama perspectives in Software AG Designer	Apama perspectives running in Software AG Designer are the main entry point for developing Apama applications.
Dashboarding tool	Software AG MashZone NextGen	MashZone NextGen's mashups provide analysis and data transformation capabilities that handle various user requirements.
Business Rules tool	webMethods Business Rules	Business Rules allows developers to create, test, and use rules that define or constrain various aspects of a business.

What is Event Routing?

webMethods Event Routing is Software AG's generic mechanism for applications to communicate using events. It plays a vital role in ensuring communication between event-enabled applications.

Event Routing offers the following capabilities:

- Easy data exchange between Software AG components
- Easy configuration using Software AG Command Central
- Reliable, fast, asynchronous data exchange between Software AG components
- Common event format supported by Software AG products

Event Routing Concepts

The Principle of Event Routing

The main principle of Event Routing is to ensure communication between the different components of the event-driven system by loose coupling of applications - one application can send events without caring which applications or services receive and consume them.

Type-Based Routing

Event Routing sends and subscribes to events using services based on event types. All events are delivered to the default service, unless one of more custom destination services are specified for

a particular event type. Event Routing subscribes for incoming events on the default service. However, you can also specify a different source service per event type. For more information about configuring Event Routing services, see "Configuring Services and Service Groups" on page 28.

Reliability

Event Routing offers a reliability setting that can be defined per service. When you set this option to true for your service, Event Routing guarantees the delivery of each event to the service. Event Routing operates in an asynchronous mode, which means that when it sends an event, it does not wait for an immediate acknowledgment of successful delivery before sending the next event. As a result, if the reliability parameter for a service is set to true, and the runtime where Event Routing is embedded crashes, any events that have been sent before the crash but have not been acknowledged will be redelivered after the restart of the runtime.

If the reliability option is set to false, the delivery of events is not guaranteed. In this case, if the runtime where Event Routing is embedded crashes, no events will be redelivered. Also, if the maximum storage capacity of the event channel is reached, the oldest undelivered events will be discarded in favor of the new events that arrive.

For more information about configuring default maximum storage capacity, see "Configuring Storage Capacity for Event Routing" on page 26. For more information about setting up storage capacity per event type, see "Configuring Event Type Associations" on page 33.

Store-and-Forward Processing

Instead of directly delivering each event to the configured destination services and waiting for each service to acknowledge the event, Event Routing stores the event in an internal queue. Depending on the reliability setting defined for each destination service via the Command Central user interface, the queue can be held in-memory or on disk. Once an event is added to the queue, Event Routing is ready to accept new events. In the meantime, the queued events are delivered to their destination services by a separate thread in the order in which they were added to the queue.

Sending Events Asynchronously

Event Routing insures a higher data throughput by providing a method for sending events asynchronously. When an asynchronous send operation is performed, Event Routing queues the event for further processing, but does not wait for the queue to be synchronized with the disk, even in case any of the destination services are set to reliable via the Command Central user interface, thus making it possible for the next send operation to be executed faster. When an asynchronous send operation is invoked, the calling application must provide a callback, which Event Routing then uses to notify the application when the event has been queued and, in case of a reliable destination service, the queue has been synchronized with the disk.

You can also send events synchronously using the respective Event Routing method in your applications. In case a synchronous send operation is performed, and the destination service is set to reliable via the Command Central user interface, Event Routing queues the event for

processing and synchronizes the queue with the disk before processing the next send operation. In this case, the throughput of events is slower.

Event Types

An event type is a schema definition that describes how events in an event stream are structured. Event types are first-class objects that are declared at a high level in the environment and can be processed by webMethods and non-webMethods products.

Events in the same stream always have the same payload structure. The schema defines which data fields are present in each event, the data type of each field, and the order in which the fields appear. Each event stream has exactly one event type associated with it. One event type can be used as the schema for more than one event stream. All event publishers on a given stream must ensure that their published events comply with the stream's schema, and all subscribers must be aware of the schema that describes the events received. In this respect, the schema represents a contract between publishers and consumers of events of a specific type.

Event types are implemented as schemas that conform to the W3C XML Schema (XSD) specification. Within the Event Type Editor, they are displayed as a hierarchy of nodes representing the content of the event. The nodes can be field nodes, composite nodes, or references to structures in other schemas. Field nodes are leaves within the node hierarchy enabling users to specify typed text strings in the XML event. Composite nodes are containers for field nodes, composite nodes, and reference nodes. At the underlying XSD level:

- The root node is invisible and is represented as a top-level element declaration with the `substitutionGroup="eda:Payload"` attribute
- Composite nodes correspond to element declarations with a complex content model
- Field nodes are element declarations with a simple type
- References refer to top-level element or type definitions in other component schemas.

You can specify a cardinality for all visible nodes, whereas the hidden root node has a fixed cardinality of `1`, denoting that a valid XML document has exactly one root element.

The XSD, as generated by the Event Type Editor, is only a subset of the full XML Schema specification. However, you can use an almost arbitrary XSD as event schema, as long as it does *not* contain the following:

- Substitution group
- Multiple child elements with the same name
- Filterable property for an element with cardinality bigger than `1`.

In order to use a custom XSD as event schema, you must do the following:

- Add the following import statement:

```
<xsd:import namespace="http://namespaces.softwareag.com/EDA/Event"
schemaLocation="Event/Envelope.xsd"/>
```

Note:
Depending on the location of the event type schema within the Event Types directory, the `schemaLocation` attribute may contain additional leading `../` steps for moving up in the directory hierarchy.

- Add the `substitutionGroup="eda:Payload"` attribute to the declaration of the element to be the root of the event XML.

Here is a section of a sample event type schema:

```
<xsd:complexType name="PartInventoryLowType">
  <xsd:annotation>
    <xsd:documentation>Report inventory low for a part</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
     <xsd:element name="Part">
       <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="ItemID" type="xsd:string" minOccurs="1"/>
          <xsd:element name="ItemName" type="xsd:string" minOccurs="0"/>
          <xsd:element name="Model" type="xsd:string" minOccurs="0" />
          <xsd:element name="Color" type="xsd:string" minOccurs="0" />
          <xsd:element name="Shape" type="xsd:string" minOccurs="0" />
        </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
     <xsd:element name="InventoryLevel" type="xsd:integer"/>
     <xsd:element name="DesiredInventoryLevel" type="xsd:integer"/>
  </xsd:sequence>
</xsd:complexType>
```

Here is a sample event instance:

```
<PartInventoryLow>
  <Part>
    <ItemID>ABC123</ItemID>
    <ItemName>Widget </ItemName>
    <Model>XYZ</Model>
    <Color>Silver</Color>
    <Shape>Oval</Shape>
  </Part>
  <InventoryLevel>58</InventoryLevel>
  <DesiredInventoryLevel>1000<DesiredInventoryLevel>
</PartInventoryLow>
```

Event Structure

Each event on the messaging bus is composed of the following parts:

- **Header**

  The following table lists the system-defined event attributes that a header contains and the respective attribute descriptions:

Event Attribute	Description
Start	Start date and time of the event.
End	Optional. End date and time of the event. The use of this field depends on how the event is being used. If the value is absent, the consumer application may set a default one, such as start time plus one millisecond.
Kind	Optional. Indicates whether the event is a new event (`Event`) or a heartbeat (`Heartbeat`). A heartbeat event indicates the temporal progress of the stream. If a value is not specified, the default is `Event`.
Type	The unique identifier of the event type. Event types use qualified names (QNames) as the mechanism for concisely identifying the particular type. The event type combines the URI and local name as a string. For example: `{http://namespaces.softwareag.com/EDA/WebM/Process/2.0}ProcessStepInstanceChange` is the event type identifier that reports changes to a process instance. **Note:** Event types without a namespace use only their local name as event identifier. For example, the noTns.xsd event type's identifier is `noTns`.
Version	Optional. The version of the event type with which the event instance is compatible. Users specify this value if they have chosen to support event type versioning. An event should not specify a version if the event type supports versioning.
CorrelationID	Optional. A unique identifier used to associate the event instance with other event instances.
EventID	Optional. A unique identifier of the event. EDA clients can distinguish between different event instances.
Priority	Optional. The priority of the event. Possible values are: ■ `Normal` (default value) ■ `High`
ProducerID	Optional. A unique identifier of the event producer.
UserID	Optional. A unique identifier of the user who emitted the event.
FormatVersion	Optional. The version of the event format. Event Routing creates automatically a value for this attribute. Check the value in the received event to see if the event body contains headers and payload. ■ If this attribute is not present in the event headers, the event body contains both headers and payload.

Event Attribute	Description
	■ If this attribute is present in the event headers, and its value is `9.0`, the event body contains only payload.
CustomHeaders	Optional. A parent header element for any user-defined headers included as sub-elements.

All messages support the same set of header fields. Header fields contain predefined values that allow clients and providers to identify and route messages. Each of the fields supports its own set and get methods for managing data; some fields are set automatically by the send and publish methods, whereas others must be set by the client. The header contains the start and end timestamp of the event.

- **Filterable Properties (optional)**

  Event Routing supports the so-called filterable properties. If you mark a field node in the event type as filterable, its value is added to the header properties of the event. For example, for the BoothDemo event shown below, if the *Producer* and the *Presenter* fields are marked as filterable, the following key-value pairs are added:

```
PulseCommon$Producer=”Event Generator”
Presenter=”dada”
```

  At run time, when events are delivered to the routing services, event consumers may apply a filter, so that only events that match certain selection criteria are consumed. These criteria can be, for example, whether the event type is a normal event or a heartbeat, or whether the value of an element from the body of the event exceeds a certain value.

  Event header elements are always added to the filterable properties with an additional prefix `$Event$` in the key. If a node in the event schema is marked as filterable, the element is added to the filterable properties when the event is published. This allows event receivers to use filterable properties based on element values.

- **Body**

  The body contains the payload of the event. The body contains the data fields of the event, as specified in the event's schema.

Here is a sample event:

```
<evt:Event xmlns:evt="http://namespaces.softwareag.com/EDA/Event>
   <evt:Header>
     <evt:Type>{http://namespaces.softwareag.com/EDA/WebM/Sample/Pulse}Pulse
     </evt:Type>
     <evt:Start>2019-05-20T16:53:46.918-06:00</evt:Start>
     <evt:End>2019-05-20T16:53:47.918-06:00</evt:End>
     <evt:Kind>Event</evt:Kind>
     <evt:EventId>0f375801-dbd4-4a46-9f70-7015deca6c80</evt:EventID>
   </evt:Header>
   <evt:Body>
     <p1:BoothDemo
      xmlns:p1="http://namespaces.softwareag.com/EDA/WebM/Sample/Pulse">
     <p1:PulseCommon>
```

```
     <p1:Producer>Event Generator</p1:Producer>
     <p1:Subject>Pulse Test Event</p1:Subject>
     <p1:Coordinates>
       <p1:Longitude>87.44988659217529</p1:Longitude>
       <p1:Latitude>83.11056319477842</p1:Latitude>
     </p1:Coordinates>
     </p1:PulseCommon>
     <p1:Presenter>dada</p1:Presenter>
     <p1:DemoTopic>Demo2</p1:DemoTopic>
     <p1:Date>2019-04-05T17:09:33.112+03:00</p1:Date>
     </p1:BoothDemo>
   </evt:Body>
</evt:Event>
```

Heartbeats

A heartbeat is a special kind of event without a payload. It indicates that the event channel on which it is being sent is active but that no payload events are currently being sent on the same channel.

The header of a heartbeat event specifies event type corresponding to the channel the heartbeat is being sent on, the start date and time of the heartbeat, and the *Kind* header field is set to `Heartbeat`.

An example of the use of heartbeats is for CEP applications, in which heartbeats can be used within a non-event detection query to determine whether the timespan in which a certain pattern did not occur has expired.

Some applications may not support heartbeats. Event receivers can suppress receiving heartbeat events by using message selection filtering on the value of the *Kind* attribute. The following message selector can be used for this purpose:

```
$Event$Kind <>'Heartbeat' or $Event$Kind is null
```

Here is a sample heartbeat event:

```
<evt:Event xmlns:evt="http://namespaces.softwareag.com/EDA/Event">
  <evt:Header>
    <evt:Type>{http://namespaces.softwareag.com/EDA/WebM/Sample/
       InventoryMgmt/1.0}PartInventoryLow</evt:Type>
    <evt:Start>2019-05-20T16:53:46.918-06:00</evt:Start>
    <evt:Kind>Heartbeat</evt:Kind>
      <evt:EventId>0f375801-dbd4-4a46-9f70-7015deca6c80</evt:EventId>
  </evt:Header>
</evt:Event>
```

Event Type Governance

You can use CentraSite to register, categorize, and govern event definitions. The Events Development perspective in Software AG Designer offers publish and unpublish functionality for the transfer of event types to and from CentraSite. You can also use CentraSite to inspect the dependencies between event type schemas and imported component schemas.

Event Type Store

The Event Type Store provides a central location per installation where predefined and user-defined event types are stored. This shared location is used by all EDA applications within the respective installation to retrieve deployed custom event types at run time.

At design time, a local copy of the predefined event types of the Event Type Store is available for reference. By default, it is located in the *Software AG_directory* /common/PredefinedEventTypes directory. You can import this directory as an existing project in Software AG Designer to inspect the event types. User-defined event types can be created using the Event Type Editor and stored in the local copy.

Event types in the local copy must be deployed to the runtime store, so that EDA applications that process an event stream can retrieve the schema definition of the event. For more information about deploying event types, see "Deploying EDA Event Types" on page 35.

Event Routing Services

Event Routing sends to and receives events from one or more services. Each service represents an instance of an external system, such as Universal Messaging, or an instance of a service within the runtime, such as an in-process event delivery service. The services are the source and target endpoints for receiving and delivering events.

Supported Service Types

Services can be of different types:

- **Universal Messaging services**

  Use services of this type send events to or receive events from a specific Universal Messaging server realm or cluster.

  For more information about configuring Universal Messaging services, see "Creating Universal Messaging Services" on page 29.

- **In-Process service**

  Use this pre-configured service to send and receive events within the same JVM.

  Note:
  Only one In-Process service exists per event type. It must be used only as `Source and Destination` and its reliability must be set to `false`.

Service Groups

Services are grouped together as a set of one or more services to which events can be sent. One of the services in a service group can be tagged as the source of events for all event types associated with this service group.

Each Event Routing runtime contains a default service group, which is sufficient for most use cases. However, you can also define a custom service group for a particular runtime.

For every custom service group you must define the following:

- A unique display name
- A set of services in the service group
- A set of event types associated with the service group.

At runtime, each event emitted by an application is delivered to one or more services based on its event type. If an event type is associated with a custom service group, the event of this event type is routed to the services within that particular service group. When an application emits events of an event type not associated with any custom service group, these events are routed to the services defined within the default service group.

For more information about configuring service groups, see "Configuring Custom Service Groups" on page 31.

Connecting to the Messaging Bus

You can connect to the messaging bus in your event-driven environment in one of the following ways:

- **Using Event Routing**

  Event Routing is a solution that enables Software AG products to communicate using events. Event Routing uses native Software AG Universal Messaging channels as endpoints. The endpoints are defined as services using the Command Central user interface. Specific event types are associated with the different services to define which events will be emitted or consumed by the specific service.

  For more information about configuring Event Routing, see "Configuring Common Properties for Event Routing" on page 26.

- **Using the EDA-related built-in services in Integration Server**

  Integration Server interacts with many Software AG products, and provides pre-configured public services for use in the event-driven system. It supports JMS connections to Software AG Universal Messaging, and it can act as an event publisher or subscriber. As a publisher, Integration Server can convert IS document types into events and publish them to the messaging bus. As a subscriber it can transform received events into IS document types.

  In addition, Integration Server:

  - Receives events from the messaging bus using JMS triggers.
  - Includes built-in services to send EDA events via Event Routing.

  The Integration Server offers a variety of bus connectivity and data transformation features, and it contains functionality that enables you to transform non-Software AG EDA event data into Software AG EDA event data. If a third party product generates events that do not conform

to the webMethods events schema, they can be converted to the webMethods event schema by using the document transformation capabilities of Integration Server. Also, Integration Server supports sending non-Software AG EDA events to the messaging bus.

For more information about using the EDA-related Integration Server built-in services, see the PDF publication *webMethods Integration Server Built-In Services Reference*.

Connecting to the Messaging Bus Using Event Routing

Event Routing is included by default in the Integration Server OSGi profile. This documentation assumes you are familiar with and have a working knowledge of OSGi implementation and architecture.

To interact with Event Routing, you should:

- Modify your runtime configuration settings, if necessary, as described in "Configuring Common Properties for Event Routing" on page 26.
- Create your custom messaging services and service groups, as described in "Configuring Services and Service Groups" on page 28.
- Modify the default event types associations, if necessary, as described in "Configuring Event Type Associations" on page 33.

Connecting to the Messaging Bus Using the EDA-Related Integration Server Built-In Services

Integration Server enables you to transform non-Software AG EDA event data into Software AG EDA event data using the `pub.event.routing:send` built-in service. Integration Server constructs an EDA event using the parameters defined in the service and sends the event to the messaging bus using Event Routing.

You can also subscribe and unsubscribe for EDA events using the `pub.event.routing:subscribe` and `pub.event.routing:unsubscribe` built-in services.

For more information about using the Event Routing-related Integration Server built-in services, see "Using the Event Routing Integration Server Built-In Services" on page 38.

For more general information about using Integration Server built-in services, see the PDF publication *webMethods Integration Server Built-In Services Reference*.

2 Administering Event Routing

This section provides information about administering Event Routing using Software AG Command Central.

Important:
To administer Event Routing within your Software AG installation, you need to have installed the Platform Manager plug-in for Event Routing, as well as the Platform Manager plug-ins for the respective products, for example Software AG Universal Messaging.

The Platform Manager plug-in for Event Routing enables you to configure routes that determine the event flow through your system, as well as specify different values for various parameters, such as Event Type Store location.

Using Command Central, you can also configure services of Universal Messaging type, group those services together, and associate event types to them. You can also configure a service within the group to serve as source and/or destination for events of a particular type.

For more information about configuring Event Routing services, see "Configuring Common Properties for Event Routing" on page 26.

For more information about using Event Routing, see "What is Event Routing?" on page 14.

Important:
In case your event-driven environment contains applications that use the JMS capabilities of Integration Server (or the JMS protocol in general) to send and receive events of a particular EDA event type, and you create applications that use Event Routing to send and receive events of the same EDA event type, the respective JNDI entries might not be created correctly in the Universal Messaging server for the two application types to work. As a workaround, open a command line prompt in *Software AG_directory* \common\lib, and run the event-routing-jms-util.jar utility. This will ensure the JNDI entries and topics have been created in your Universal Messaging server. For more information about how to run the utility, see the readme file in event-routing-jms-util.jar.

Configuring Common Properties for Event Routing

Important:
To administer Event Routing within your Software AG installation, you need to have installed the Platform Manager plug-in for Event Routing, as well as the Platform Manager plug-ins for the respective products, for example Software AG Universal Messaging.

This section contains information about common properties available for Event Routing as a runtime component in your Software AG installation.

For information about configuring services and service groups, see "Configuring Services and Service Groups" on page 28.

Configuring Storage Capacity for Event Routing

Depending on the reliability configured for each service within a service group, all events sent to Event Routing are stored on disk or in-memory.

For more information about configuring services, see "Creating Universal Messaging Services" on page 29 and "Configuring Custom Service Groups" on page 31.

Important:
You need to stop the runtime where Event Routing is embedded before you modify the value of the **Storage Location** property. If Event Routing is running in a Platform Manager runtime, you must restart Platform Manager for the changes to take effect. In case the previous storage location is already in use, you must copy any existing files manually to the new storage location before modifying the value of the **Storage Location** property.

To configure the storage capacity for Event Routing

1. In Command Central, navigate to **Environments > Instances > All > <*profile_name*> > Event Routing > Configuration** tab, select **Runtime Configuration** from the drop-down menu, and then click **Edit**.

2. Specify values for the fields in the following table as explained in the description column:

Field	Description
Storage Location	Defines the location where events are stored on disk. The provided value must be an existing folder. The default value is *Software AG_directory* \profiles*profile_name*\configuration\event\routing\runtime\storage, where *profile_name* is the name of the OSGi profile where Event Routing is embedded. If left blank, the default value applies. If your product runs outside an OSGi environment, the value of this property corresponds to the value you defined using the *com.softwareag.event.routing.configuration.directory* system property. **Note:** You need to stop the runtime where Event Routing is embedded before you modify the value of this property.
Default On-Disk Capacity	Defines the maximum number of events of a particular event type stored on disk. The default value is `1000000`.
Default In-Memory Capacity	Defines the maximum number of events of a particular event type stored in the memory. The default value is `1000`.

3. Save your changes.

Event Routing detects that the configuration has been updated, and starts to use the new settings automatically.

Setting up System Properties for Products Using Event Routing Outside the Software AG Common Platform

When you embed Event Routing in a product that runs outside the Software AG Common Platform, you must configure several system properties beforehand.

To use Event Routing outside the Common Platform, set up the system properties in the following table as explained in the description column:

System property	Definition
com.softwareag.event.routing.configuration.directory	
	Specifies the path to the Event Routing configuration directory.
com.softwareag.event.routing.eventtypestore.location	
	Specifies the path to the Event Type Store location.
com.softwareag.event.routing.security.file.location	
	Specifies the Event Routing security file location. This file contains the encrypted secret key used by Event Routing for encrypting and decrypting passwords specified in the Event Routing services.

Configuring Services and Service Groups

Services are grouped together as a set of one or more services to which events can be sent. One of the services in a service group can be tagged as the source of events for all event types associated with this service group.

Each Event Routing runtime contains a default service group, which is sufficient for most use cases. However, you can also define a custom service group for a particular runtime.

For more information about the types of services you can configure, see “Supported Service Types” on page 21.

For more conceptual information about service groups, see “Service Groups” on page 21.

Important:
To administer Event Routing within your Software AG installation, you need to have installed the Platform Manager plug-in for Event Routing, as well as the Platform Manager plug-ins for the respective products, for example webMethods Integration Server or Software AG Universal Messaging.

Creating Universal Messaging Services

You can create and configure services of Universal Messaging type, add them to groups, and associate event types to them.

To create and configure Event Routing services of type Universal Messaging

1. In Command Central, navigate to **Environments > Instances > All > *instance_name* > Event Routing > Configuration** tab.

2. Select **Messaging Services** from the drop-down menu, and then click .

3. Specify values for the fields in the following table as explained in the description column:

Field	Description
Service Name	Required. A unique name for the service. The service name is not case-sensitive, and must start with a character. You can use the following characters as separators: . (dot) and - (dash).
Description	Required. A description of the purpose of the service.
Provider URL	Required. Supports Secure Sockets Layer (SSL). The host and port of the Universal Messaging server to which the service connects. The default value is `nsp://localhost:9000`. You can use an SSL-enabled Universal Messaging server, for example: `nsps://localhost:9000`.
User Name	Optional. The name of the user authorized to connect to a Universal Messaging server with server-side authentication enabled.
Password	Optional. The password of the user authorized to connect to a Universal Messaging server with server-side authentication enabled.

Important:
You cannot rename a service which has already been created. If you wish to modify a service name, you must delete the old one and recreate a new one with a different name.

4. Save your changes.

Event Routing detects that the configuration has been updated, and starts to use the new settings automatically.

Using Universal Messaging Services with SSL

Event Routing supports the Universal Messaging NSPS protocol for secure communication.

To enable Event Routing to communicate with an SSL-enabled Universal Messaging server, you must configure certain Java system properties for the runtime in which Event Routing is running.

The following table lists the Java system properties and the values you specify:

Property	Value
javax.net.ssl.keyStore	The path to the client keystore.
javax.net.ssl.keyStorePassword	The password for the client keystore.
javax.net.ssl.trustStore	The path to the certificate authority (CA) keystore file.
javax.net.ssl.trustStorePassword	The password for the CA keystore.

Note:
Event Routing supports only .jks file based keystore and truststore types.

For more information about configuring Universal Messaging for SSL communication, see the Universal Messaging documentation. For more information about configuring Java system properties, see *Software AG Command Central Help* or the documentation of the embedding runtime.

Configuring the Default Service Group

Each product runtime where Event Routing is used has a default service group. All event types that are not explicitly associated to any custom service group are considered associated to the default one. This means that any events from those event types are delivered to the services within the default service group.

Note:
You cannot rename or delete the default service group.

You can modify the default behavior of Event Routing by adding or removing services from the default service group.

Note:
You must have created and configured the services before adding them to the default service group.

For more information about creating services, see "Creating Universal Messaging Services" on page 29.

To configure the default service group Event Routing

1. In Command Central, navigate to **Environments > Instances > All > *instance_name* > Event Routing> Configuration** tab, select **Service Groups** from the drop-down menu, and then click **Default** in the **Service Group Name** column.

2. To add existing services to the group, click  and select them from the drop-down menu in the **Service Name** field.

3. For each service you add to the default service group, define the properties in the following table as explained in the description column:

Property	Description
Reliable	Defines whether Event Routing guarantees the delivery of every event to the service. By default this value is set to `true`. Possible values are `true` and `false`. **Note:** You can set this property to `true` only for services whose type supports reliable delivery of events.
Usage	Defines whether the service serves as a source and/or destination for events. By default this value is set to `Destination Only`.Possible values are: ■ `Source Only` - the service only consumes events. Only supported for services that can act as a source. ■ `Destination Only` - the service only emits events. ■ `Source and Destination` - the service emits and consumes events. Only supported for services that can act as a source. **Note:** Your service group must contain only one source service. You cannot include the same service twice in the same service group. If you selected the predefined In-Process service, it must always be set to `Source and Destination`.

4. Optionally, click **Test** to verify your configuration is consistent.

5. Save your changes.

Event Routing detects that the configuration has been updated, and starts to use the new settings automatically.

Configuring Custom Service Groups

After installing Event Routing, you have one default service group that contains the preconfigured **UniversalMessaging** service. You can create one or more custom service groups and associate a set of event types to them. When events of those particular event types are sent or received, they go to all services within the service group. One of the services in the group can be marked as source and/or destination of events for all event types associated to the service group.

Note:
You must have created and defined at least one service before creating a custom service group.

For more information about creating services, see "Creating Universal Messaging Services" on page 29.

To configure custom service groups for Event Routing

1. In Command Central, navigate to **Environments > Instances > All > *profile_name* > Event Routing > Configuration** tab, select **Service Groups** from the drop-down menu, and then click  .

2. In the **Configuration Details** dialog, enter a name and a description for the new service group.

   Note:
   The service group name must be unique. It is not case-sensitive, and must start with a character. You can use the following characters as separators: . (dot) and - (dash).

   Note:
   You cannot rename a service group which has already been created. If you wish to modify a service group name, you must delete the old one and recreate a new one with a different name.

3. To add existing services to the group, click  and select them from the drop-down menu in the **Service Name** field.

4. For each service you add to the new service group, define the properties in the following table as explained in the description column:

Property	Description
Reliable	Defines whether Event Routing guarantees the delivery of every event to the service. By default this value is set to `true`. Possible values are `true` and `false`. **Note:** You can set this property to `true` only for services whose type supports reliable delivery of events.
Usage	Defines whether the service serves as a source and/or destination for events. By default this value is set to `Destination Only`.Possible values are: ■ `Source Only` - the service only consumes events. Only supported for services that can act as a source. ■ `Destination Only` - the service only emits events.

Property	Description
	■ `Source and Destination` - the service emits and consumes events. Only supported for services that can act as a source. **Note:** Your service group must contain only one source service. You cannot include the same service twice in the same service group. If you selected the predefined In-Process service, it must always be set to `Source and Destination`.

5. Optionally, click **Test** to verify your configuration is consistent.

6. Save your changes.

Event Routing detects that the configuration has been updated, and starts to use the new settings automatically.

Configuring Event Type Associations

You can associate a set of predefined and custom event types that exist in your Event Type Store to different service groups. When events of those particular event types are sent or received, Event Routing delivers them to all services within the respective service groups.

Note:
All event types, which are not explicitly associated to a custom service group, are associated to the default service group.

Note:
You must have already defined your custom service groups before associating event types to them. You cannot delete a custom service group which has event types associated to it.

For more information about defining service groups, see "Configuring Services and Service Groups" on page 28.

To configure event type associations

1. In Command Central, navigate to **Environments > Instances > All > *profile_name* > Event Routing > Configuration** tab, select **Event Type Associations** from the drop-down menu, and then click the **Event Type Associations** entry.

2. In the **Configuration Details** dialog, click **Edit**.

3. In the **Service Group** column, select the service group from the drop-down menu to associate the respective event type to it.

4. In the **In-Memory Capacity** and **On-Disk Capacity** columns, enter the desired values to set the maximum number of events stored in the Store and Forward queue for a particular event type.

   You can use any positive integer or specify `1K` (1024), `1M` (1024K) or `1G` (1024M). You can leave an empty string or enter `Default` to denote usage of the default global settings.

   For more information about setting up common storage capacity settings, see "Configuring Storage Capacity for Event Routing" on page 26.

5. Optionally, click **Test** to verify your configuration is consistent.

6. Save your changes.

Event Routing detects that the configuration has been updated, and starts to use the new settings automatically.

Securing Passwords Held in Service Configurations

In specific cases when you create Event Routing service configurations making connections which require password authentications, you must provide authentication credentials as values within the service configurations. The passwords are encrypted in all service configuration files using a secret key, which is also encrypted.

The secret key is used by Event Routing to encrypt and decrypt passwords specified in the service configurations making connections which require password authentications. The secret key can be found in the event-routing-security.xml file available in the *Software AG_directory* /common/conf/event/routing directory.

Software AG recommends that you modify the value of the secret key on each Software AG installation containing an Event Routing node before you start configuring services in Command Central. You can use the Event Routing ciphering utility to encrypt a given value and use it as a secret key.

For more information about using the Event Routing ciphering utility, see "Working with the Event Routing Ciphering Utility" on page 34.

For more information about modifying the secret key, see "Modifying the Event Routing Secret Key" on page 35.

Working with the Event Routing Ciphering Utility

The Event Routing ciphering utility enables you to encrypt a given value and use it as a secret key. You can then use the new encrypted value to replace the secret key used by Event Routing to encrypt and decrypt passwords specified in your service definitions. The Event Routing ciphering utility is delivered as part of your Software AG installation and can be found as a .jar file in the *Software AG_directory* /common/conf/event/routing directory.

To use the Event Routing ciphering utility

1. In a command prompt, navigate to *Software AG_directory* /common/conf/event/routing.

2. Execute the `java -jar` command and specify the nerv-cipher-util.jar, as well as a value to be encrypted, for example:

```
java -jar <Software_AG_Directory>/common/lib/nerv-cipher-util.jar
<value_to_be_encrypted>
```

The encrypted value is displayed in the command prompt and c;an be used as a secret key for encrypting and decrypting user credentials provided to Event Routing by custom route bundles making connections which require password authentications.

Modifying the Event Routing Secret Key

By default, Event Routing is delivered with an encrypted secret key which is used for encrypting and decrypting user credentials provided by custom route bundles making connections which require password authentications. Software AG recommends that you modify the value of the secret key on each Software AG installation containing an Event Routing node before you start developing your Event Routing custom route bundles.

To modify the Event Routing secret key

1. In your file system, navigate to the nerv-security.xml file available in the *Software AG_directory* /common/conf/event/routing directory.

   The file contains the default encrypted secret key value:

```
<nervSecurity>
      <key>{AES}9bexKOp6S06Y8IJL53b4P8wCXf3pKWBrI8/vbOqlnhA=</key>
</nervSecurity>
```

2. Using the Event Routing ciphering utility, generate a new encrypted value for the secret key.

   For more information about the Event Routing ciphering utility, see "Working with the Event Routing Ciphering Utility" on page 34.

3. For each Software AG installation containing a product which embeds Event Routing, use the newly generated value to replace the default secret key in the in the nerv-security.xml file available in the *Software AG_directory* /common/conf/event/routing directory.

4. Recreate any existing service definition that which contains a password.

Event Routing uses the new secret key to encrypt and decrypt passwords which are part of service definitions.

Deploying EDA Event Types

Deployment is the process of moving EDA assets from the design environment into the run-time or production environment.

EDA event types composites can be deployed to one or more target runtimes using the webMethods Deployer's repository-based deployment. To use this deployment method, you must have the Asset Build Environment (ABE) installed.

The EDA event types composites that you create prior to deployment must have a specific structure in order to be deployable using Deployer. Event Types deployment composites are valid Event Types projects - a parent project directory with an Event Types subdirectory containing event type definitions. Software AG recommends that you use Software AG Designer's Events Development perspective to develop your custom event type definitions.

The event type definitions are considered individual assets and are packed by the Asset Build Environment into zip archives. Multiple event type definitions can be packed in a single zip file.

Note:
Event type schemata with namespaces that do not start with the `http://namespaces.softwareag.com/EDA` string are deployed to the WebM/External directory of the Event Type Store.

When you enable the creation of EDA composites and run the ABE build script, the script searches the specified source directories and creates a composite for each project directory that contains EDA event types.

For more information about installing the Asset Build Environment feature, see *Installing Software AG Products*. For more information about building composites for repository-based deployment, see *webMethods Deployer User's Guide*.

Example of a Deployment Project Structure

You can use Asset Build Environment to build deployable composites from EDA event types. Here is an example of an EDA source repository directory and the deployable assets which are produced by the Asset Build Environment build script. In the example below the *build.source.dir* property is set to `/source` as a prerequisite.

For Event Types with the following source repository structure:

```
/source/MyNewEvents/Event Types/MyCompany/Account.xsd
/source/MyNewEvents/Event Types/MyCompany/Receipt.xsd
```

the Asset Build Environment build script creates the *MyNewEvents.zip* deployable composite, which contains the two Account and Receipt event types.

3 Event Routing for Developers

If you are a developer working in an event-enabled environment, you can create custom applications using the capabilities offered by Event Routing.

Using the Event Routing Integration Server Built-In Services

Integration Server interacts with many Software AG products, and provides pre-configured public services for use in the event-driven system. It supports JMS connections to Software AG Universal Messaging, and it can act as an event publisher or subscriber. As a publisher, Integration Server can convert IS document types into events and publish them to the messaging bus. As a subscriber it can transform received events into IS document types.

The pre-configured Event Routing built-in services are available in the WmPublic\pub\event\routing folder of your Integration Server packages.

Using the pub.event.routing:send Service

The procedure below explains how to use the `pub.event.routing:send` service to send events. It assumes that you are familiar with working with built-in services and flow services in Software AG Designer. For more information about IS built-in services, see the PDF publication *webMethods Integration Server Built-In Services Reference*. For more information about working with flow services, see the PDF publication *webMethods Service Development Help*.

To send EDA events using the `pub.event.routing:send` service

1. In the Service Development perspective in Designer, create a new document type from an existing event type, for example the PartInventoryLow event type.

   You can use any event type that has previously been deployed to the Event Type Store in your Software AG installation. For more information about deploying event types, see "Deploying EDA Event Types" on page 35.

   a. Use the PartInventoryLow event name as a name for the new document type and click **Next**.

   b. Select XML Schema as source type and click **Next**.

   c. Select **File/URL** for source location, and browse to the PartInventoryLow event type in the Event Type Store and click **Next**.

      By default, the PartInventoryLow event type is located in the *Software AG_directory* \common\EventTypeStore\WebM\Sample\InventoryMngt\1.0 directory.

   d. On the next page of the wizard, leave the schema-related processing options unchanged, and click **Next**.

   e. Select the *PartInventoryLow* element as the root node, enable the **Expand complex type inline** option for a cleaner layout, then click **Next**.

f. On the next page of the wizard you can configure the namespace prefixes to be used for representing namespaces found in the schema. Leave the entries unmodified, and click **Finish**.

2. Create a new empty flow service.

3. In the Input/Output tab of the Flow service editor, in the Input Parameters panel, insert a document reference to the new document type you created in step 1.

   Note:
   You can drag and drop the document type from Package Navigator view.

4. In the Tree or the Layout tab of the Flow service editor, insert an INVOKE pub.event.routing:send step.

5. In the Pipeline view, link the document reference from the Pipeline Input area to the event/body node of the pub.event.routing:send service in the Service Input area.

6. In the Service Input area, set the value of the *Type* variable to the full event type name, in this example `{http://namespaces.softwareag.com/EDA/WebM/Sample/InventoryMgmt/1.0}PartInventoryLow`.

7. In the Service Input area, set the value of the *documentTypeName* variable to refer to the document type you created in step 1.

   This is required in order to assert that the namespace declarations are added to the XML document emitted as an EDA event.

8. Right click and select **Run As > Run Flow Service** to test your flow service.

   Note:
   The *name* attribute of the second `<record>` element must match the name of the document reference configured as the input of the flow service.

Using the pub.event.routing:subscribe Service

The procedure below explains how to use the `pub.event.routing:subscribe` service to subscribe to events. It assumes that you are familiar with working with built-in services and flow services in Software AG Designer. For more information about IS built-in services, see the PDF publication *webMethods Integration Server Built-In Services Reference*. For more information about working with flow services, see the PDF publication *webMethods Service Development Help*.

To subscribe to EDA events using the `pub.event.routing:subscribe` service

1. In the Service Development perspective in Designer, create a new empty flow service, and open it.

2. In the Tree tab of the Flow service editor, drag and drop the pub.event.routing:subscribe service.

3. In the Pipeline view, set the *eventTypeName* to the name of the desired event type, in this example `{http://namespaces.softwareag.com/EDA/WebM/Sample/InventoryMgmt/1.0}PartInventoryLow`, and the *serviceName* to the name of a service that will be invoked when an event of the specified type is received.

   Note:
   The service specified by the*serviceName* parameter must have a document reference to the `pub.event.eda:event` service. Other input parameters are not allowed.

4. Right click and select **Run As > Run Flow Service** to subscribe to events using the flow service.

Note:
You can configure the newly created flow service as a startup service. For more information about startup services, see *webMethods Service Development Help*.

Using the pub.event.routing.unsubscribe Service

The procedure below explains how to use the `pub.event.routing:unsubscribe` service to unsubscribe from events. It assumes that you are familiar with working with built-in services and flow services in Software AG Designer. For more information about IS built-in services, see the PDF publication *webMethods Integration Server Built-In Services Reference*. For more information about working with flow services, see the PDF publication *webMethods Service Development Help*.

To unsubscribe from EDA events using the `pub.event.routing:unsubscribe` service

1. In the Service Development perspective in Designer, create a new empty flow service, and open it.

2. In the Tree tab of the Flow service editor, drag and drop the pub.event.routing:unsubscribe service.

3. In the Pipeline view, set the *eventTypeName* and the *serviceName* to the same values that were specified in the subscription service.

4. Right click and select **Run As > Run Flow Service** to subscribe to events using the flow service.

Note:
You can configure the newly created flow service as a shut down service. For more information about shut down services, see *webMethods Service Development Help*.

Developing Applications for Configuring Event Routing

Event Routing configuration is integrated with Software AG Command Central. You can configure Event Routing programmatically in a specified runtime using your custom applications. This can be achieved by using the Command Central command line interface or by using the Command Central REST API.

For more information, see the following sections below:

- "Using Command Central Composite Templates to Configure Event Routing" on page 46.
- "Monitoring Event Routing Data" on page 49.

Using the Command Central Command Line Interface for Configuring Event Routing

You can use the Command Central command line interface in your custom applications to configure your Event Routing runtime component.

Below you can find a list of most relevant Event Routing-related commands, and examples of their usage in the context of Event Routing. Note that the example syntax is as required for execution in Command Central.

For the full list of available Command Central commands and their options, as well as syntax examples for execution in Software AG Platform Manager, see *Software AG Command Central Help*.

The following table lists the most relevant Event Routing-related commands and their descriptions:

Related Command	Description
`sagcc list configuration types`	Gets all available configuration types for a particular runtime component. For a usage example, see "Getting all Available Event Routing Configuration Types" on page 42.
`sagcc list configuration instances`	Gets a list of all objects of a particular configuration type. For a usage example, see "Getting a List of all Objects of a Particular Event Routing Configuration Type" on page 42.
`sagcc get configuration data`	Gets configuration data for a specified object. For a usage example, see "Getting Data for a Specific Configuration Instance" on page 42.
`sagcc update configuration data`	Updates a configuration object. For a usage example, see "Updating Specific Configuration Data" on page 42.
`sagcc create configuration data`	Adds a configuration object. For a usage example, see "Adding Configuration Data" on page 43.
`sagcc delete configuration data`	Deletes a configuration object. For a usage example, see "Deleting Configuration Data" on page 43.

Getting all Available Event Routing Configuration Types

You can retrieve information about all available Event Routing configuration types for a specified Event Routing instance running within a product's OSGi profile. Use the following command:

```
sagcc list configuration types node_alias componentid [typeid] [options]
```

For example, to get all configuration types available for an Event Routing component running within the Integration Server OSGi profile on an installation with alias "sag01", execute the following command:

```
sagcc list configuration types sag01 OSGI-IS_default-EventRouting
```

Getting a List of all Objects of a Particular Event Routing Configuration Type

You can get a list of all objects of a particular Event Routing configuration type with an ID, name, display name, and description. Use this command together with the `sagcc get configuration data` command to retrieve detailed information for each object. Use the following command:

```
sagcc list configuration instances node_alias componentid [instanceid] [options]
```

For example, to get a list of all configuration instances of an Event Routing component running within the Integration Server OSGi profile on an installation with alias "sag01", execute the following command:

```
sagcc list configuration instances sag01 OSGI-IS_default-EventRouting
```

Getting Data for a Specific Configuration Instance

You can get data about a specified configuration instance that belongs to a specified Event Routing component. Use the following command:

```
sagcc get configuration data node_alias componentid instanceid [options]
```

For example, to get configuration data for all Universal Messaging services in the Event Routing component running within the Integration Server OSGi profile on an installation with alias "sag01", execute the following command:

```
sagcc get configuration data sag01 OSGI-IS_default-EventRouting UniversalMessaging
```

Updating Specific Configuration Data

You can update data about a specified configuration instance of a specified Event Routing component. Use the following command:

```
sagcc update configuration data node_alias componentid instanceid
[sharedsecret=text_string ]# {--input | -i}
filename {.xml|.json|.properties} [options]
```

For example, to update the configuration data for a Universal Messaging service with the name "UM1" in the Event Routing component running within the Integration Server OSGi profile on an installation with alias "sag01", execute the following command:

```
sagcc update configuration data sag01
OSGI-IS_default-EventRouting COMMON-WMMESSAGING-ER
UM1 --input C:\ConfigStore\UM1.json
```

Where “C:\ConfigStore\UM1.json” is a file containing the updated configuration for the “UM1” service.

Note:
The data in the input file must match the expected format for the configuration type. You can use the `sagcc exec configuration validation update` command to validate the input data that you want to use for updating your configuration instance.

Adding Configuration Data

You can create a new instance of a specified configuration type for a specified Event Routing component. Use the following command:

```
sagcc create configuration data node_alias componentid typeid {--input | -i}
file {.xml|.json|.properties} [options]
```

For example, to create a new instance of Universal Messaging service type with the name “UM1” in the Event Routing component running within the Integration Server OSGi profile on an installation with alias “sag01”, execute the following command:

```
sagcc create configuration data sag01 OSGI-IS_default-EventRouting
COMMON-WMMESSAGING-ER --input C:\ConfigStore\UM1.json
```

Where “C:\ConfigStore\UM1.json” is a file containing the configuration for the new “UM1” service.

Deleting Configuration Data

You can delete a configuration instance from a specified Event Routing component. Use the following command:

```
sagcc delete configuration data node_alias componentid instanceid [options]
```

For example, to delete the “UM1” messaging service of Universal Messaging service type in the Event Routing component running within the Integration Server OSGi profile on an installation with alias “sag01”, execute the following command:

```
sagcc delete configuration data sag01 OSGI-IS_default-EventRouting
COMMON-WMMESSAGING-ER UM1
```

Note:
The restrictions for deleting services and service groups using the Command Central web user interface are also applicable when using the command line interface.

Configuration Types that Event Routing Supports

The following table lists the configuration types that the Event Routing run-time component supports and their descriptions:

Configuration Type	Description
RUNTIME-CONFIGURATION	Runtime configuration settings for Event Routing. For information about the fields and values to specify when configuring Event Routing runtime settings, see "Configuring Common Properties for Event Routing" on page 26.
SERVICE-GROUP	Groups of services for a particular runtime. One of the services in a service group can be tagged as the source of events for all event types associated with this service group. For information about Event Routing service groups, see "Configuring Services and Service Groups" on page 28.
COMMON-WMMESSAGING-ER	Services of Universal Messaging type that are the source or target endpoints for receiving and delivering events. For information about Event Routing services, see "Configuring Services and Service Groups" on page 28.
EVENT-TYPE-ASSOCIATIONS	Associations of predefined and custom event types that exist in the Event Type Store to different Event Routing service groups. For information about Event Routing event type associations, see "Configuring Event Type Associations" on page 33.

For general information about using Command Central to configure settings for a product, see *Software AG Command Central Help*.

Using the Command Central REST API for Configuring Event Routing

You can use the Command Central REST API in your custom applications to configure Event Routing.

For more information about using the Command Central REST API, see the *Getting Started with the Software AG Command Central REST API* PDF publication.

For a list of commands that can be called using the Command Central REST API, see "Using the Command Central Command Line Interface for Configuring Event Routing " on page 41.

Examples for Using the Command Central REST API for Configuring Event Routing

In order to be able to configure Event Routing using the Command Central REST API, you should be familiar with the landscape, inventory, and configuration services of Command Central.

To get detailed information about which REST URLs are supported by the services, and how they can be used with the HTTP methods GET, PUT, POST, and DELETE, issue the following calls:

- http://*cc-host*:*cc-port*/cce/landscape/application.wadl
- http://*cc-host*:*cc-port*/cce/inventory/application.wadl
- http://*cc-host*:*cc-port*/cce/configuration/application.wadl

Where *cc-host* is the name of the host machine where you have installed Command Central, and *cc-port* is the port number where the Command Central instance is running.

The following sample REST URLs assume that your Command Central server can be reached under `localhost:8090`, where `localhost` is the node alias of the default installation.

- `GET http://localhost:8090/cce/landscape/nodes` - gets all nodes (and their nodeAliases) in the landscape.
- `GET http://localhost:8090/cce/inventory/components` - gets all runtime components in the landscape. Component IDs for Event Routing always end with “EventRouting”. For example, “OSGI-SPM-EventRouting” is the ID of the Event Routing component installed inside Software AG Platform Manager.
- `GET http://localhost:8090/cce/configuration/instances/{nodeAlias}/{runtimeComponentId}` - gets all metadata for all configuration instances of a specific component on the defined node. For example,

  ```
  GET http://localhost:8090/cce/configuration/instances/local/
  OSGI-SPM-EventRouting
  ```

  gets metadata for all configuration instances of the “OSGI-SPM-EventRouting” component on the “local” node. This metadata contains the configuration ID which is required in other REST calls.
- `GET http://localhost:8090/cce/configuration/data/{nodeAlias}/{runtimeComponentId}/{configurationInstanceId}` - gets the actual configuration data for a configuration instance in a specified runtime component. For example,

  ```
  GET http://localhost:8090/cce/configuration/data/local/
  OSGI-SPM-EventRouting/UniversalMessaging
  ```

  gets the actual configuration data of the “UniversalMessaging” configuration of the “OSGI-SPM-EventRouting” component. You can use PUT with the same REST URL to update the configuration instance. You can use DELETE with the same REST URL to delete the configuration instance.
- `GET http://localhost:8090/cce/configuration/types/{nodeAlias}/{runtimeComponentId}` - gets all configuration types that can be used with a specified runtime component. For example,

  ```
  GET http://localhost:8090/cce/configuration/types/local/
  OSGI-SPM-EventRouting
  ```

gets all configuration types that can be used with the “OSGI-SPM-EventRouting” component.

The Event Routing component has the following configuration types: `RUNTIME-CONFIGURATION`, `SERVICE-GROUP`, `COMMON-WMMESSAGING-ER`, `EVENT-TYPE-ASSOCIATIONS`.

- `POST http://localhost:8090/cce/configuration/data/{nodeAlias}/{runtimeComponentId}/{configurationTypeId}` - creates a new configuration object of the specified configuration type of the component. For example,

```
POST http://localhost:8090/cce/configuration/data/local/
OSGI-SPM-EventRouting/SERVICE-GROUP
```

creates a new configuration object of type “SERVICE-GROUP” of the “OSGI-SPM-EventRouting” component.

Using Command Central Composite Templates to Configure Event Routing

With Command Central, you can create new environments using composite templates. A composite template defines a set of environment properties for which you can specify values or use the default values provided in the template definition. For example, you can specify which products to include in a composite template or the number of nodes on which to apply the template.

With Command Central, you import and apply composite templates using the Command Central command line tool.

For more information about composite templates and the related command line commands, see *Software AG Command Central Help*.

Configuring Event Routing in a Single Runtime

You can provision Event Routing configuration in a single runtime instance on a specified installation. For example, in an Integration Server profile on your installation, you can define a Universal Messaging service called “MyUniversalMessaging”, and configure the default service group to use it as a source and destination service. To do this, define the following Event Routing template:

```
templates:
  is-esb:
    # ...
  er-config:
    instance:
     productId: integrationServer
     name: ${is.instance.name}    # You can also fix the name such as
                                  IS_default in the template directly
     configuration:
      *-EventRouting:
        COMMON-WMMESSAGING-ER:
          COMMON-JMS-umdefault:
            alias: MyUniversalMessaging,
            enabled: true,
            URL: nsp://${umhost}:${umport}
```

```
          service-groups:
            name: default
            description: The default event group includes all event types
                         that are not added to other defined groups
            services:
              service-name: MyUniversalMessaging,
              service-is-reliable: true
              service-usage: SourceAndDestination
```

Note:
The example above contains variables that use the following format: *${parameterName}*. In case you use variables instead of hard-coded values, make sure you create a properties file containing values for those variables, and provide it when applying your composite templates.

Now add the newly created Event Routing template to all layers which include product instances where you want to provision the Event Routing configuration:

```
layers:
  esb:
    description: Enterprise Service Bus layer based on Integration Server
    templates:
      is-esb
      er-config
  bpm:
    description: Business Process Management layer based on Integration Server
    templates:
      is-bpm
      er-config
```

Configuring Event Routing in Multiple Installations

You can provision Event Routing configuration in all runtime instances on multiple installations. For example, in all Integration Server instances on several installations, you can define a Universal Messaging service called “MyUniversalMessaging”, and configure the default service group to use it as a source and destination service. To do this, you must:

- Define the installations in your composite template

  Add the following code snippet:

```
nodes:
  default:
   default:
     port: 8093
     secure: true
     credentials:
       username: Administrator
       password: manage
  envType1:
   node1:
     host: localhost
     port: 8192
   node2:
     host: localhost
     port: 8292
```

- Define the layers and list the templates that should be applied

  Add the following code snippet:

```
layers:
  esb:
   templates:
    is-esb
    er-config
  bpm:
   templates:
    is-bpm
    er-config
```

- Define the environments

  Add the following code snippet:

```
environments:
   envType1:
     param2: someValue
   envType2:
     param1: v1
     param3: v3
```

- Map the layers to the nodes.

  This is done separately for each environment. The example below shows that the nodes for the `envType2` environment are not hard-coded directly in the template, but are resolved by using input parameters.

```
provision:
   envType1:
     bpm: [node1,node2]
     esb: [node2,node3]
   envType2:
     bpm: ${bpm.hosts}
     esb: ${is.hosts}
```

- Define the following Event Routing template that applies to all product instances in the installation.

```
templates:
  is-esb:
   # ...
  is-bpm:
   # ...
  er-config:
    instance:
      productId: *   # Use * to have unified Event Routing configuration
             across the installation, you can be selective if necessary
      name: *        # Use * again (any instance)
      configuration:
       *-EventRouting:
         COMMON-WMMESSAGING-ER:
           COMMON-JMS-umdefault:
             alias: MyUniversalMessaging,
             enabled: true,
             URL: nsp://${umhost}:${umport}
```

```
        service-groups:
          name: default
          description: The default event group includes all event types
                       that are not added to other defined groups
          services:
            service-name: MyUniversalMessaging,
            service-is-reliable: true
            service-usage: SourceAndDestination
```

Note:
The example above contains variables that use the following format: *${parameterName}*. In case you use variables instead of hard-coded values, make sure you create a properties file containing values for those variables, and provide it when applying your composite templates.

Applying Composite Templates

The composite templates can be applied to your environment using command line commands. For the full list of available command line commands, see *Software AG Command Central Help*.

- To apply a composite template that uses hard-coded values, in a command prompt, run the following command:

```
cc exec templates composite apply mytemplate param1=value1 param2=value2
```

- To apply a composite template where values for variables are provided in a .properties file:

```
cc exec templates composite apply mytemplate -i myparams1.properties
```

Monitoring Event Routing Data

As a developer of a product that uses Event Routing, you can create a JMX client to retrieve configuration information exposed by Event Routing. This enables you to monitor Event Routing data flowing through your system.

As a developer creating Event Routing services, you can expose configuration information specific to your service.

Note:
You must not expose any service-specific attributes of complex types in your JMX client. You must only use generic Java types, such as *String*, *Integer/int*, *Long/br_long*, *Date*, etc.

Common Attributes for All Service MBeans

The following table lists the names of the attributes that are common for all service Mbeans and the respective attribute descriptions:

Name	Description
usage	String. The usage of the Event Routing service. Possible values are: - `Source only` - you can subscribe to services of this type, but you cannot send events to them. - `Destination only` - you can send events to services of this type, but you cannot subscribe to them. - `Source and Destination` - you can send events and subscribe to services of this type.
reliability	String. The reliability setting of the service. Possible values are: - `reliable` - `best-effort`
status	String. The status of the service after an event was sent through it to the destination server or another type of endpoint. The value of this attribute is updated every time an event is sent from the queue to the destination service or another endpoint, depending on whether the server/ endpoint has acknowledged the event with success or failure. Possible values are: - `green` when the service is functioning correctly. - `red` when the service is unavailable. **Note:** If the destination server or endpoint becomes unavailable, but no events have been sent for a long time, the status will still be `green`. In case you are monitoring a Universal Messaging service, use this parameter in conjunction with connected, connectTime, and disconnectTime to determine whether the Universal Messaging server is available.
statusDetails	String. An explanation of the current status. Possible values are: - `green` when the service is functioning correctly. - `red` when the service is unavailable. Provides the message of the last error which occurred while trying to deliver events to the destination server or another type of endpoint denoted by the service.

Name	Description
receivedEvents	Long. The number of events received by all subscribers to this service since its activation. If the service usage is `Destination only`, the value of this attribute will be empty. Otherwise, the value will be: ■ Growing at a steady pace, when the service is functioning correctly. ■ Growing slowly, when the service is slow. ■ Not growing for a long period of time, when the service is unavailable. **Note:** In case you are monitoring a Universal Messaging service, use this parameter in conjunction with connected, connectTime, and disconnectTime to determine whether the Universal Messaging server is available.
activeDurableSubscribersCount	Integer. The number of active durable subscribers to this service. If the service usage is `Destination only`, the value of this attribute will be empty.
activeDurableSubscribersIds	List<String>. The identifiers of the active durable subscribers to this service. If the service usage is `Destination only`, the value of this attribute will be empty.
inactiveDurableSubscribersCount	Integer. The number of inactive (closed but still subscribed) durable subscribers to this service. If the service usage is `Destination only`, the value of this attribute will be empty.
inactiveDurableSubscribersIds	List<String>. The identifiers of the inactive (closed but still subscribed) durable subscribers to this service. If the service usage is `Destination only`, the value of this attribute will be empty.
nonDurableSubscribersCount	Integer. The number of non-durable subscribers to this service. If the service usage is `Destination only`, the value of this attribute will be empty.
sentEvents	Long. The number of events sent to the service. If the service usage is `Source only`, the value of this attribute will be empty. The value of this attribute is: ■ Growing if the service is functioning correctly.

Name	Description
	■ Growing up to the point when the on-disk queue is full, if the service is reliable. This is most probably due to a slow or unavailable service.
acknowledgedSentEvents	Long. The number of events sent through this service and successfully acknowledged by the messaging server, database, or another type of endpoint it represents. If the service usage is `Source only`, the value of this attribute will be empty. Otherwise, the value will be: ■ Growing at a steady pace, when the service is functioning correctly. ■ Growing slowly, when the service is slow. ■ Not growing for a long period of time, when the service is unavailable.
lastSendingTime	Instant. The time of sending the last event to the messaging server, database, or another type of endpoint it represents. If the service usage is `Source only`, the value of this attribute will be empty. Otherwise, the value will be: ■ A few milliseconds or seconds before the current time, when the service is functioning correctly or when the service is slow. ■ Before the current time, when the service is unavailable. **Note:** This reading denotes the time an event was last sent to the destination service. It does not contain information about whether or when the event was acknowledged by the service. For such information, see the lastCompletedAcknowledgementTime and lastFailedAcknowledgementTime attributes.
lastCompletedAcknowledgementTime	Instant. The last time an event sent through this service was acknowledged with success by the destination server. If the service usage is `Source only`, the value of this attribute will be empty. Otherwise, the value will be: ■ A few milliseconds or seconds before the current time and shortly after lastSendingTime, when the service is functioning correctly.

Name	Description
	■ Before lastSendingTime, when the service is slow or unavailable.
lastFailedAcknowledgementTime	Instant. The last time an event sent through this service was acknowledged with failure by the destination server. If the service usage is `Source only`, the value of this attribute will be empty. Otherwise, the value will be: ■ `empty` or pointing to a long time before the current time, when the service is functioning correctly. ■ `empty` or pointing to a recent moment in the past, when the service is slow. The delivery of an event might fail due to a slow connection to the destination server, in which case the event must be redelivered. ■ After lastSendingTime and a few milliseconds or seconds before the current time (depending on the redelivery interval and the response timeout of the service), when the service is unavailable.
regularDeliveryInterruptionsCount	Integer. The number of times the service has switched to redelivery mode because of failure to deliver an event to the destination server or another type of endpoint. The delivery of an event might fail due to a slow connection to the destination server, in which case the event must be redelivered. If the service usage is `Source only`, the value of this attribute will be empty.

Attributes for Universal Messaging Service MBeans

The following table lists the names of the attributes for the Universal Messaging Service Mbeans and the respective attribute descriptions:

Name	Description
providerUrl	String. The URL of the Universal Messaging server in the following format: *<protocol>://<host>:<port>*, for example: `nsp://localhost:9000`
connected	Boolean. Indicates whether the Universal Messaging service is connected to the configured Universal Messaging server. Possible values are:

Name	Description
	■ `true` when the service is functioning correctly or is slow. ■ `false` when the service is disconnected.
connectTime	Instant. The time when the Universal Messaging service connected to the configured Universal Messaging server, or reconnected to it after a disconnection. Possible values are: ■ Before the current time when the service is functioning correctly or is slow. ■ Empty when the service is initially unavailable. ■ Before the time represented by disconnectTime when the Universal Messaging service becomes unavailable afterwards.
disconnectTime	Instant. The time when the Universal Messaging service disconnected from the Universal Messaging server. Possible values are: ■ Empty or before the value represented by connectTime when the Universal Messaging service is functioning correctly or is slow. ■ Empty when the Universal Messaging service is initially unavailable. ■ After the value represented by connectTime when the Universal Messaging service becomes unavailable afterwards.

Attributes for Queue MBean

The following table lists the names of the attributes for the Queue Mbean and the respective attribute descriptions:

Name	Description
status	String. The status of the queue. Possible values are: ■ `green` when the queue is functioning correctly. ■ `yellow` when the queue is slow.

Name	Description
	■ `red` when the queue is unavailable. This might be caused by the destination server being down for a longer period of time.
statusDetails	String. An explanation of the current status: ■ `green` when the queue's utilization is below 90%. ■ `yellow` when the queue's utilization is equal to or greater than 90% but less than 100%. ■ `red` when the queue is full; new events will be rejected in case of an on-disk queue, or the oldest events will be discarded when new events arrive in case of an in-memory queue.
capacity	Long. The maximum number of event which the queue can store.
currentSize	Long. The number of events currently stored in the queue. Possible values are: ■ `green` when the queue capacity is below 90%. ■ `yellow` when the queue capacity is equal to or greater than 90%. ■ `red` when the queue capacity is equal to 100%.
averageUtilization	Float. The average size:capacity ratio since the first send operation for the event type which uses the queue.
queueBufferFullCount	Integer. The number of times the queue has become full, that is currentSize has been equal to capacity. Possible values are: ■ `0` - if the queue status has never been red. ■ `> 0` - when the queue status is red now, or has been red before.