Informatica® Edge Data Streaming
2.5.0

Developer Guide

Informatica Edge Data Streaming Developer Guide
2.5.0
July 2019

Publication Date: 2019-07-16

Table of Contents

Preface

The Informatica Edge Data Streaming Developer Guide is written for developers who want to develop custom entities for Edge Data Streaming. The Developer Guide provides information about the APIs available and how to use them to develop custom entities.

The Developer Guide assumes that you are familiar with the Java programming language and APIs.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

Informatica maintains documentation for many products on the Informatica Knowledge Base in addition to the Documentation Portal. If you cannot find documentation for your product or product version on the Documentation Portal, search the Knowledge Base at https://search.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

Introduction to Edge Data Streaming Custom Entity Types

This chapter includes the following topics:

Edge Data Streaming Custom Entity Types Overview

Informatica Edge Data Streaming (Edge Data Streaming) is a distributed, extensible, and scalable data aggregation solution. You can use Edge Data Streaming to move large volumes of high-velocity data from multiple points of generation to one or more data targets in real time.

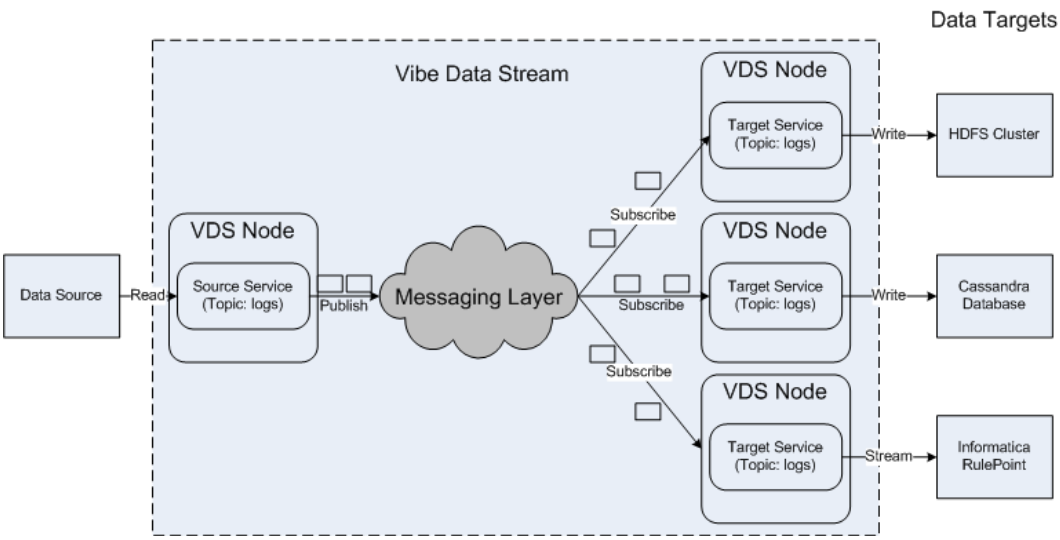
To transfer data, you use source services and target services. Source services read and publish data in the form of messages on a particular topic. Target services that subscribe to the topic receive the messages and write the messages to the target. Source services and target services run on Edge Data Streaming nodes.

You create the nodes on the data source hosts and data target hosts. You can also create the nodes on hosts that have access to the data sources and data targets. In Informatica Administrator (the Administrator tool), you create a data flow design by adding source services and target services to a data flow. You connect the two services, and you add any transformations that you want to apply.

You map the source service and target service to the Edge Data Streaming nodes on which they must run. When you deploy the data flow design, the Edge Data Streaming node to which you mapped the source service starts the source service. The Edge Data Streaming node to which you mapped the target service starts the target service. The source service collects data from the data source in real time. The target service receives and writes the data to the data target.

You can add multiple source services, multiple target services, and multiple transformations to a data flow.

The following image shows the flow of messages from a source service to multiple target services through transformations:



In the Administrator tool, you add source services, target services, or transformations to a data flow by adding appropriate entity types. For example, to receive and write messages to a Hadoop Distributed File System (HDFS) cluster, you use the built-in HDFS target service type.

Use built-in entity types for common use cases. Use built-in source service types for aggregating data from flat files, TCP sources, UDP sources, Syslog sources, and MQTT brokers. Use built-in target service types for writing data to HDFS, Apache Cassandra, Informatica PowerCenter®, Informatica RulePoint, and Informatica Ultra Messaging. Use a built-in regular expression filter and JavaScript and string insert transformation types to transform data.

If an entity type that you want is not available in Edge Data Streaming, create a custom entity type by using the Edge Data Streaming application programming interfaces (APIs). Use the APIs to create custom source service types, custom target service types, and custom transformation types.

Edge Data Streaming API

You can use the EDS Java APIs to develop custom source service types, custom target service types, and custom transformation types. The EDS Java API consists of interfaces that you implement to create the custom entity types. The API package includes helper classes that you can use to perform common operations in the interface implementation.

You can use also the Maven archetypes to develop custom source service types, custom target service types, and custom transformation types. The Maven archetypes are packaged with the EDS APIs. You can use these archetypes to create custom service types in an Eclipse environment.

Informatica recommends that you use the Maven archetypes to create entities.

You can use the Representational State Transfer (REST) API to create, retrieve, update, or delete EDS entities directly over HTTP. You can also deploy, redeploy, and undeploy EDS entities.The HTTP request methods supported are GET and POST. The sample rest calls show you how to use the REST API with application/json type.

Edge Data Streaming API Installation Package

The EDS APIs are available in the `EDS-api-reference.zip` ZIP file that is available with the EDS installation package. Download and extract the ZIP file to a directory on your machine.

The `EDS-api-reference.zip` ZIP file contains the following folders:

archetypes

The `archetypes` folder contains the following Maven archetypes you can use to create entities:

- `vds-template-source-archetype`. The archetype for the source service.
- `vds-template-target-archetype`. The archetype for the target service.
- `vds-template-transform-archetype`. The archetype for the transformation.

catalog

The `catalog` folder contains the `archetype-catalog.xml` file which stores the archetype information.

docs

The `docs` folder contains the API reference documentation. To view the API reference documentation, open the file `index.html` file.

lib

The `lib` folder contains the `vdsapi-2.5.jar` EDS Java API JAR file.

schema

The `schema` folder contains the `eds_plugin.xsd` EDS plug-in schema definition that describes the format of the EDS plug-in XML file.

scripts

The `scripts` folder contains the following API installer files:

- `install-eds-api.bat` . The Windows installer file.
- `install-eds-api.sh`. The Linux installer file.

Edge Data Streaming Interfaces

EDS includes four interfaces, one each to create a custom source service type, custom target service type, custom transformation service type, and statistics. All the EDS interfaces except the VDSPluginStatistics interface and the VDSConsumptionSourceextend interface extend the `Closeable` Java interface.

You can use the following five EDS interfaces:

VDSSource

The EDS interface that you implement to create a source service type. The interface includes methods to initialize the source, read data from the data source, and close the source.

VDSConsumptionSource

The EDS interface that you implement to create a source service type that requires acknowledgment from the target service after the message has been consumed. The interface includes methods to initialize the source, read data from the data source, and close the source.

VDSTarget

The EDS interface that you implement to create a target service type. The interface includes methods to initialize the target, write data to the target, and close the target.

VDSTransform

The EDS interface that you implement to create a transformation type. The interface includes methods to initialize the transformation, transform the data, and close the transformation.

VDSPluginStatistics

The EDS interface that you implement to create entity statistics. The interface includes the method to get statistics.

Edge Data Streaming Helper Classes

EDS includes a collection of helper classes for common operations that you need to perform in your EDS interface implementation.

EDS includes the following helper classes:

VDSConfiguration

A `VDSConfiguration` object stores the configuration of an entity as a set of key-value pairs. The `VDSConfiguration` helper class includes methods to retrieve the value of a key and a method to check for the presence of a key.

VDSErrorCode

The `VDSErrorCode` class contains all the error codes that EDS uses.

VDSEvent

A `VDSEvent` object is an event and contains a block of data that EDS is transferring from the source location to the target location. An event is a message.

VDSEventList

The `VDSEventList` class represents a list of events in EDS. You can add events that sources and transformations generate to a `VDSEventList` object.

VDSException

The `VDSException` class includes details about an exception in EDS.

CHAPTER 2

Managing Custom Entity Types

This chapter includes the following topics:

Managing Custom Entity Types Overview

Create the components that a custom entity type requires and add the components to a ZIP file that you can use in EDS. Version the entity type so that you can upgrade or downgrade the entity type.

You can use also the Maven archetypes to develop custom source service types, custom target service types, and custom transformation types. The Maven archetypes are packaged with the EDS APIs. You can use these archetypes to create custom service types in an Eclipse environment.

Informatica recommends that you use the Maven archetypes to create entities.

Components of a Custom Entity Type

Organize the components of the custom entity type by using a particular directory structure.

A custom entity type consists of the following components:

EDS Plug-in XML File (Required)

An XML file called `vdsplugin.xml`. The XML file describes configuration and plug-in details for the custom entity type.

EDS Plug-in JAR File (Required)

The EDS plug-in JAR file contains the implementation of a VDS interface.

Plug-in Dependant Java Class Libraries (Optional)

The Java class libraries are libraries on which the implementation of the custom entity type depends. Store the Java class libraries in a directory called `lib`. The lib directory can contain multiple JAR files.

Plug-in Dependant Native Libraries (Optional)

Native libraries are shared objects (`.so` files) for Linux and dynamically loaded library (`DLL` files) for Windows. Native libraries are required if the Java libraries have native dependencies. Store the native libraries in a directory called `native`. The native directory can contain multiple .so or DLL files.

Create all the components in the same directory. For example, to create the components in a directory called `plugin`, use the following directory structure:

```
plugin
  |- vdsplugin.xml                  VDS plug-in XML file
  |- <MyVDSPlugin>.jar              VDS plug-in JAR file
  +- lib                            Directory that contains Java class libraries
  |   |- <JavaClassLibrary1>.jar
  |   |- <JavaClassLibrary2>.jar
  +- native                         Directory that contains native libraries
  |   |- <SharedObject1>.so
  |   |- <SharedObject2>.so
```

Note: The angle brackets (< and >) indicate that you can replace the name in the example with a name of your choice.

Creating a Custom Entity Type

To create a custom entity type, perform the following tasks:

- Create the EDS plug-in XML document.
- Create the EDS plug-in JAR file.
- Add dependent libraries. Place the Java class libraries on which your implementation depends in a directory called `lib`. Place native libraries in a directory called `native`.
- Create a package for the custom entity type. Package the EDS plug-in XML document, the EDS plug-in JAR file, and the `lib` and `native` directories in a ZIP file.

Creating the Edge Data Streaming Plug-in XML Document

Create an XML document called `vdsplugin.xml`. In the XML document, describe the configuration of the custom entity type.

You can create the XML document in the following ways:

Create the EDS Plug-in XML Document Manually

Manually edit an XML file to include all the required XML elements.

Generate a Sample XML Document from the EDS Plug-in Schema and Modify the XML Document

Use a tool of your choice to generate an XML document from the schema definition.

Perform the following tasks:

- Create a copy of the `field` element for each field that you want the entity to contain, and then provide values for the XML elements for each field. For example, in the version element, specify a version number.

  Note: Specify a unique plug-in ID in the XML document. The plug-in ID must not contain spaces. If two or more EDS plug-in JAR files have the same plug-in ID, EDS loads only one of those plug-in JAR files, and only that plug-in JAR file is visible as a custom entity type in the Administrator tool. For the plug-in JAR files that EDS does not load, EDS logs an error.

- Edit the values of the XML elements that the `vdsPlugin` XML element encloses.

After you create the EDS plug-in XML document, validate the document against the schema definition.

Format of the Edge Data Streaming Plug-in XML Document

The EDS plug-in XML document consists of a set of XML elements that describe a custom entity type. The file is `vdsplugin.xml`.

The following XML document shows a EDS plug-in XML document with fields that you use:

```
<tns:vdsPlugin>
    <tns:id>AllSrc</tns:id>
    <tns:displayName>All Source Types</tns:displayName>
    <tns:version>3.0</tns:version>
    <tns:type>SOURCE</tns:type>
    <tns:configuration>
        <tns:fields>
            <tns:field>
                <tns:textControl>
                    <tns:name>txt</tns:name>
                     <tns:displayName>Text Field</tns:displayName>
                     <tns:description>Enter value for text field</tns:description>
                      <tns:mandatory>true</tns:mandatory>
                      <tns:stringTextField>
                          <tns:pattern>^[A-Z]*$</tns:pattern>
                          <tns:maxLength>30</tns:maxLength>
                          <tns:secure>false</tns:secure>
                        <tns:default>ABCD</tns:default>
                        <tns:placeholder>A-Z</tns:placeholder>
                      </tns:stringTextField>
                </tns:textControl>
            </tns:field>
            <tns:field>
                <tns:textControl>
                    <tns:name>pwd</tns:name>
                     <tns:displayName>Password</tns:displayName>
                     <tns:description>Enter value for password</tns:description>
                      <tns:mandatory>true</tns:mandatory>
                      <tns:stringTextField>
                          <tns:maxLength>10</tns:maxLength>
                          <tns:secure>true</tns:secure>
                      </tns:stringTextField>
                </tns:textControl>
            </tns:field>
            <tns:field>
                <tns:textControl>
                    <tns:name>int</tns:name>
                     <tns:displayName>Int Field</tns:displayName>
                     <tns:description>Enter value for int field</tns:description>
                      <tns:mandatory>true</tns:mandatory>
                      <tns:integerTextField>
                          <tns:minValue>10</tns:minValue>
                          <tns:maxValue>90</tns:maxValue>
                        <tns:default>30</tns:default>
                      </tns:integerTextField>
                </tns:textControl>
            </tns:field>
            <tns:field>
```

```
                    <tns:textControl>
                        <tns:name>dou</tns:name>
                         <tns:displayName>Double Field</tns:displayName>
                         <tns:description>Enter value for double field</tns:description>
                          <tns:mandatory>false</tns:mandatory>
                          <tns:doubleTextField>
                              <tns:minValue>11.11</tns:minValue>
                              <tns:maxValue>99.99</tns:maxValue>
                            <tns:default>33.33</tns:default>
                          </tns:doubleTextField>
                    </tns:textControl>
                </tns:field>
                <tns:field>
                    <tns:radioGroupControl>
                        <tns:name>radio</tns:name>
                          <tns:displayName>Mother Tongue</tns:displayName>
                          <tns:description>Enter value for mother tongue</tns:description>
                          <tns:items>
                              <tns:item>
                                <tns:displayName>English</tns:displayName>
                                   <tns:id>1</tns:id>
                              </tns:item>
                              <tns:item>
                                  <tns:displayName>Hindi</tns:displayName>
                                  <tns:id>2</tns:id>
                                <tns:textFields>
                                    <tns:textControl>
                                          <tns:name>txtScript</tns:name>
                                          <tns:displayName>Script</tns:displayName>
                                          <tns:mandatory/>
                                          <tns:stringTextField>
                                            <tns:secure/>
                                          </tns:stringTextField>
                                      </tns:textControl>
                                    <tns:textControl>
                                          <tns:name>txtUnicode</tns:name>
                                          <tns:displayName>Unicode</tns:displayName>
                                          <tns:mandatory/>
                                          <tns:stringTextField>
                                            <tns:secure/>
                                          </tns:stringTextField>
                                       </tns:textControl>
                                </tns:textFields>
                                <tns:subListBoxField>
                                    <tns:listControl>
                                          <tns:name>variant</tns:name>
                                          <tns:displayName>Variant</tns:displayName>
                                          <tns:items>
                                              <tns:item>
                                                  <tns:displayName>Bihar</tns:displayName>
                                                <tns:id>1</tns:id>
                                              </tns:item>
                                              <tns:item>
                                                  <tns:displayName>Rajasthan</
tns:displayName>
                                                <tns:id>2</tns:id>
                                              </tns:item>
                                            <tns:item>
                                                <tns:displayName>U.P.</tns:displayName>
                                                <tns:id>3</tns:id>
                                            </tns:item>
                                         </tns:items>
                                         <tns:default>2</tns:default>
                                      </tns:listControl>
                                </tns:subListBoxField>
                              </tns:item>
                          </tns:items>
                          <tns:default>1</tns:default>
                      </tns:radioGroupControl>
                </tns:field>
                <tns:field>
```

```
                    <tns:listControl>
                        <tns:name>list</tns:name>
                          <tns:displayName>Qualifications</tns:displayName>
                          <tns:description>Enter value for qualifications</tns:description>
                          <tns:items>
                              <tns:item>
                                <tns:displayName>Degree</tns:displayName>
                                <tns:id>1</tns:id>
                              </tns:item>
                              <tns:item>
                                <tns:displayName>Masters</tns:displayName>
                                <tns:id>2</tns:id>
                                <tns:textFields>
                                    <tns:textControl>
                                        <tns:name>txtField1</tns:name>
                                        <tns:displayName>College</tns:displayName>
                                        <tns:mandatory>false</tns:mandatory>
                                        <tns:stringTextField>
                                            <tns:secure/>
                                          </tns:stringTextField>
                                    </tns:textControl>
                                </tns:textFields>
                                <tns:subListBoxField>
                                    <tns:listControl>
                                          <tns:name>division</tns:name>
                                          <tns:displayName>Division</tns:displayName>
                                          <tns:items>
                                              <tns:item>
                                                  <tns:displayName>A1</tns:displayName>
                                                <tns:id>1</tns:id>
                                              </tns:item>
                                              <tns:item>
                                                  <tns:displayName>A2</tns:displayName>
                                                <tns:id>2</tns:id>
                                              </tns:item>
                                            <tns:item>
                                                <tns:displayName>A3</tns:displayName>
                                                <tns:id>3</tns:id>
                                            </tns:item>
                                          </tns:items>
                                          <tns:default>3</tns:default>
                                       </tns:listControl>
                                </tns:subListBoxField>
                            </tns:item>
                            <tns:item>
                                <tns:displayName>Ph.D</tns:displayName>
                                <tns:id>3</tns:id>
                            </tns:item>
                        </tns:items>
                          <tns:default>3</tns:default>
                      </tns:listControl>
                </tns:field>
                <tns:field>
                    <tns:checkBoxControl>
                        <tns:name>chkBox</tns:name>
                          <tns:displayName>Married</tns:displayName>
                          <tns:description>Marital Status</tns:description>
                          <tns:default/>
                          <tns:checkedFields>
                              <tns:textControl>
                                <tns:name>txtFieldM1</tns:name>
                                <tns:displayName>Spouse</
tns:displayName>
                                <tns:mandatory/>
                                <tns:stringTextField>
                                    <tns:secure/>
                                </tns:stringTextField>
                               </tns:textControl>
                          </tns:checkedFields>
                          <tns:uncheckedFields>
                              <tns:textControl>
```

```
                            <tns:name>txtFieldM2</tns:name>
                            <tns:displayName>Mother</
tns:displayName>
                            <tns:mandatory/>
                            <tns:stringTextField>
                                <tns:secure/>
                            </tns:stringTextField>
                           </tns:textControl>
                      </tns:uncheckedFields>
                </tns:checkBoxControl>
            </tns:field>
            <tns:field>
                <tns:textAreaControl>
                    <tns:name>txtArea</tns:name>
                      <tns:displayName>Comments</tns:displayName>
                      <tns:description>Your comments</tns:description>
                      <tns:mandatory>true</tns:mandatory>
                      <tns:maxLength>30</tns:maxLength>
                      <tns:default></tns:default>
                      <tns:placeholder>Place Src holder</tns:placeholder>
                </tns:textAreaControl>
            </tns:field>
        </tns:fields>
    </tns:configuration>
    <tns:pluginStatistics>
                    <tns:statistic>
                                    <tns:id>1</tns:id>
            <tns:displayName>Statistic1</tns:displayName>
            <tns:type>CUMULATIVE</tns:type>
        </tns:statistic>
        <tns:statistic>
            <tns:id>2</tns:id>
            <tns:displayName>Statistic2</tns:displayName>
            <tns:type>CUMULATIVE</tns:type>
        </tns:statistic>
    </tns:pluginStatistics>
    <tns:runTime>
        <tns:pluginJar>AllSource.jar</tns:pluginJar>
        <tns:pluginClass>com.informatica.vds.plugin.custom.AllTypeSource</
tns:pluginClass>
    </tns:runTime>
  <tns:helpKey></tns:helpKey>
</tns:vdsPlugin>
```

The XML document consists of the following XML elements:

vdsPlugin

Encloses all the XML elements that describe the custom entity type.

id

The EDS plug-in ID. The value must be unique across the plug-ins that you use in the EDS deployment. The value must not contain spaces and can have a maximum length of 32 characters.

displayName

A name for the entity. EDS displays the entity type with the specified name in the **Entity Types** pane of the Administrator tool. `_vds_entity_name` is a reserved word and must not be used as the name of any field.

For example, if you use the preceding XML document, EDS displays the entity type with the name `MyCustomSource`.

version

A string used to version the entity type.

For example, if you use the preceding XML document, EDS creates an entity type with the version number `1.0`.

type

The type of entity that you want EDS to create. The type can be `SOURCE`, `TARGET`, or `TRANSFORMATION`.

For example, if you use the preceding XML document, EDS creates a source service.

configuration

Describes the properties of the fields in the dialog box that a user uses to configure the entity in the Administrator tool. The `configuration` element encloses the following element:

fields

Describes the fields in the dialog box. The `fields` element encloses the following element:

field

Encloses the XML elements that describe the field. You can create multiple `field` elements within the `fields` element, each with its own set of properties.

The `field` element encloses the following element:

textControl

Encloses the XML elements that describe the field type and properties.

The `textControl` element encloses the following elements:

name

Specifies an internal name for the field. Use the internal name to reference the entity in your implementation of a EDS interface. `_VDS_ENTITY_NAME` is a reserved word and must not be used as the name of any field.

displayName

Specifies the name with which the field must appear when a user adds the entity to a data flow.

For example, if the field accepts a directory path for the source file, use the name `directory`.

description

An internal description for the field.

mandatory

Makes the field required or optional. The setting `true` indicates that the field is required. The setting `false` indicates that the field is optional.

stringTextField

Specifies that the field is a string. The `stringTextField` element encloses the following elements:

- `pattern`. Specifies the pattern that text field can hold.
- `maxLength`. Specifies the maximum length of the text field.
- `secure`. Specifies that the field is secure.

  Note: If you have installed EDS in secure mode, the field is securely communicated using encryption over UM or HTTPS, and then encrypted and stored in the database. If you have installed EDS in a normal mode, the value in the field appears masked in the Administrator tool, and is not encrypted.
- `default`. Specifies the default value of the field

- `placeholder`. Specifies the placeholder text that must appear when a user adds the entity to a data flow.

integerTextField

Specifies that the field is an integer. The `integerTextField` element encloses the following elements:

- `minValue`. Specifies the minimum value that the field can hold.
- `maxValue`. Specifies the maximum value that the field can hold.
- `default`. Specifies the default value of the field.

doubleTextField

Specifies that the field is a double text field. The `doubleTextField` element encloses the following elements:

- `minValue`. Specifies the minimum value that the field can hold.
- `maxValue`. Specifies the maximum value that the field can hold.
- `default`. Specifies the default value of the field.

radioGroupControl

Specifies that the field is an option button.

The `radioGroupControl` element encloses the following elements:

name

Specifies the name of the option button.

displayName

Specifies the name with which the option button must appear when you add the entity to the data flow.

description

Specifies the description of the option button.

items

Specifies the options that are part of the option button. This element encloses the `item` element. If you require to specify the conditional fields you can specify them within the `item` element. Each `item` element encloses the `textControl` and the `listControl` elements. You can add multiple `item` elements.

default

Specifies the option that should be selected the default.

listControl

Specifies that the field is a drop-down list field.

The `listControl` element encloses the following elements:

name

Specifies the name of the drop-down list field.

displayName

Specifies the name with which the drop-down list field must appear when you add the entity to the data flow.

description

Specifies the description of the drop-down list field.

items

Specifies the options that are part of the option button. This element encloses the `item` element. If you require to specify the conditional fields you can specify them within the `item` element. Each `item` element encloses the `textControl` and the `listControl` elements. You can add multiple `item` elements.

default

Specifies the option that should be selected the default.

checkBoxControl

Specifies that the field is a checkbox field.

The `checkBoxControl` element encloses the following elements:

name

Specifies the name of the checkbox field.

displayName

Specifies the name with which the checkbox field must appear when you add the entity to the data flow.

description

Specifies the description of the checkbox field.

default

Specifies the field that is select by default.

checkedFields

Specifies the list of fields that are selected. If you require to specify the conditional fields you can specify them within the `checkedFields` element. This element encloses the `textControl` element.

uncheckedFields

Specifies the list of fields that are not selected. This element encloses the `textControl` and the `listControl` elements.

textAreaControl

Specifies that the field is a text area field.

The `textAreaControl` element encloses the following elements:

name

Specifies the name of the text area.

displayName

Specifies the name with which the text area must appear when you add the entity to the data flow.

description

Specifies the description of the text area.

mandatory

The setting `true` indicates that the field is required. The setting `false` indicates that the field is optional.

maxLength

Specifies the maximum length of the text area.

default

Specifies the default value of the text area.

placeholder

Specifics the placeholder text that must appear when you add the text area to the data flow.

pluginStatistics

Encloses information about the statistics that you want to create. The `pluginStatistics` element encloses the following element:

statistic

Encloses information that describe the statistics that you create. The `statistic` encloses the following elements:

id

The statistic ID. The value must be unique across the plug-ins that you use in the EDS deployment. The value must not contain spaces and can have a maximum length of 32 characters.

displayName

Specifies the name with which the statistic must appear when a user adds it to the entity.

type

The type of statistic that you want EDS to create. The type can be CUMULATIVE.

runTime

Encloses information that EDS needs to run the entity at run time. The `runTime` element encloses the following elements:

pluginJar

The name of the JAR file that contains the implementation of a EDS interface.

For example, if you use the preceding XML document, EDS creates the entity by using the plug-in `mySourcePlugin.jar`

pluginClass

The fully qualified name of the class that implements a EDS interface. For example, if you name the class that implements a EDS interface `myClass`, and you add the class to the package `com.myOrg.myPackage`, enter `com.myOrg.myPackage.myClass`.

Sample Implementations of Edge Data Streaming Interfaces

Create a Java source file that implements the `VDSSource`, `VDSTarget`, or `VDSTransform` interface. For information about the EDS API, see the EDS API reference documentation that the EDS Java API includes. The following code samples show how to implement the EDS interfaces.

Implementing the VDSSource Interface

Implement the `VDSSource` interface to create a custom source service type.

The following sample code shows how to implement the `VDSSource` interface:

```
package ...;

import java.io.IOException;
import com.informatica.vds.api.VDSConfiguration;
import com.informatica.vds.api.VDSEventList;
import com.informatica.vds.api.VDSSource;

public class CustomSource implements VDSSource {

    /* Initialize the VDSSource object. */
    @Override
    public void open(VDSConfiguration vdsconfiguration) throws Exception {
        /* The internal name of the key. */
        String fieldName = "SourceURLFromUI";

        /* The default value of the key. */
        String defaultValue = "SourceURLDefault";

        /* Retrieve the value of the key SourceURLFromUI. */
        String SourceURL = vdsconfiguration.optString("SourceURLFromUI",
"SourceURLDefault");

        /* Similarly, retrieve other keys. */
        ...
    }

    /* Read the data from the data source. */
    @Override
    public void read(VDSEventList vdsevents) throws Exception {
        /* Collect data into a byte array called data. */
        ...

        /* Add the generated data to the VDSEventList object. */
        vdsevents.addEvent(data, data.length);
    }

    @Override
    public void close() throws IOException {
    /* Clean up. */
    ...
    }

                @Override
       public void setRetryPolicyHandler(IPluginRetryPolicy arg0) {
              // TODO Auto-generated method stub

    }

}
```

The following sample code shows how to implement the `VDSSource` interface to shard events:

```
package ...;

import java.io.IOException;
import com.informatica.vds.api.VDSConfiguration;
import com.informatica.vds.api.VDSEventList;
```

```
import com.informatica.vds.api.VDSSource;

public class CustomSource implements VDSSource {

    /* Initialize the VDSSource object. */
    @Override
    public void open(VDSConfiguration vdsconfiguration) throws Exception {
        /* The internal name of the key. */
        String fieldName = "SourceURLFromUI";

        /* The default value of the key. */
        String defaultValue = "SourceURLDefault";

        /* Retrieve the value of the key SourceURLFromUI. */
        String SourceURL = vdsconfiguration.optString("SourceURLFromUI",
"SourceURLDefault");

        /* Similarly, retrieve other keys. */
        ...
    }

    /* Read the data from the data source. */
                @Override
       public void read(VDSEventList arg0) throws Exception {
           System.out.println("Reading source...");
        Thread.sleep(10);
        String message = (String) queue.poll();
        if (message == null) {
            return;
        }
        count++;
        Map<String, String> map = new HashMap();


        if (count % 2 == 0) {
            map.put("transportTopic", "TwoTopic");

        } else if (count % 3 == 0) {
            map.put("transportTopic", "ThreeTopic");

        } else {
            map.put("transportTopic", "OtherTopic");
        }

        arg0.addEvent(message.getBytes(), message.getBytes().length, map);

        if (count % 2 == 0) {
                 statValues.put((short)2, ++twos);
              }
              if (count % 3 == 0) {
                     statValues.put((short)3, ++threes);
              }

              System.out.println("Data " + message.getBytes() + " " +
message.getBytes().length);
               }
             @Override
                public void close() throws IOException {
                /* Clean up. */
                ...
                        }

                @Override
       public void setRetryPolicyHandler(IPluginRetryPolicy arg0) {
              // TODO Auto-generated method stub

                           }
}
```

Implementing the VDSConsumptionSource Interface

Implement the `VDSConsumptionSource` interface to create a custom source service type.

The following sample code shows how to implement the `VDSConsumptionSource` interface:

```
package ...;

import java.io.IOException;

import com.informatica.vds.api.VDSConfiguration;
import com.informatica.vds.api.VDSConsumptionSource;
import com.informatica.vds.api.VDSEventList;

public class SampleConsumptionSource implements VDSSource, VDSConsumptionSource {

    @Override
    public void open(VDSConfiguration ctx) throws Exception {
        // TODO Auto-generated method stub

    }

    @Override
    public void read(VDSEventList readEvents) throws Exception {
        // TODO Auto-generated method stub

    }

    @Override
    public void close() throws IOException {
        // TODO Auto-generated method stub

    }

    @Override
    public void onVDSEventConsume(Object obj) throws Exception {
        // TODO Auto-generated method stub

    }

}
```

Implementing the VDSTarget Interface

Implement the `VDSTarget` interface to create a custom target service type.

The following sample code shows how to implement the `VDSTarget` interface:

```
package ...;

import java.io.IOException;
import java.nio.ByteBuffer;
import com.informatica.vds.api.VDSConfiguration;
import com.informatica.vds.api.VDSEvent;
import com.informatica.vds.api.VDSTarget;

public class CustomTarget implements VDSTarget {

    @Override
    public void close() throws IOException {
     /* Close the connection to the target and clean up. */
     ...
    }

    /* Initialize the VDSTarget object. */
    @Override
    public void open(VDSConfiguration vdsconfiguration) throws Exception {
        /* The internal name of the key. */
        String fieldName = "TargetURLFromUI";
```

```
        /* The default value for the key. */
        String defaultValue = "TargetURLDefault";

        /* Retrieve the value of the key TargetURLFromUI. */
        String TargetURL = vdsconfiguration.optString(fieldName, defaultValue );

        /* Similarly, retrieve other keys. */
        ...

        /* Create or initialize the target by using the configuration
         * information.
         */
        ...
    }

    /* Get data and write the data to the target. */
    @Override
    public void write(VDSEvent vdsevent) throws Exception {
        /* Get data and collect it in a byte array. */
        ByteBuffer bytebuff = vdsevent.getBuffer();
        byte[] bytearr = new byte[vdsevent.getBufferLen()];
        bytebuff.get(bytearr);

        /* Write the data to the target. */
         ...
    }
            @Override
            public void setRetryPolicyHandler(IPluginRetryPolicy arg0) {
            // TODO Auto-generated method stub
            }
}
```

Implementing the VDSTransform Interface

Implement the `VDSTransform` interface to create a custom transformation type.

The following sample code shows how to implement the `VDSTransform` interface:

```
package ...;

import java.io.IOException;
import java.nio.ByteBuffer;
import com.informatica.vds.api.VDSConfiguration;
import com.informatica.vds.api.VDSEvent;
import com.informatica.vds.api.VDSEventList;
import com.informatica.vds.api.VDSTransform;

public class CustomTransform implements VDSTransform {

    @Override
    public void close() throws IOException {
        /* Clean up. */
        ...
    }

    /* Apply a transformation to the data. */
    @Override
    public void apply(VDSEvent vdsevent, VDSEventList vdsevents) throws Exception {
        /* Get data from the VDSEvent object. */
        ByteBuffer bytebuff = vdsevent.getBuffer();
        byte[] totdata = new byte[vdsevent.getBufferLen()];
        bytebuff.get(totdata);

        /* Transform the data. */
         ...

        /* Assume that the transformation returns a byte[] that is stored in a
         * variable named transformedData. Add transformedData to the event
         * list.
```

```
         */
         vdsevents.addEvent(transformedData, transformedData.length);
    }

    /* Initialize the VDSTarget object. */
    @Override
    public void open(VDSConfiguration vdsconfiguration) throws Exception {

        /* The internal name of the key. */
        String exprVal = "ExpressionFromUI";

        /* The default value of the key. */
        String exprDefault = "DefaultValue"

        /* Retrieve the value of the key ExpressionFromUI. */
        String Expr = vdsconfiguration.optString(exprVal, exprDefault);

        /* Similarly, retrieve other keys. */
        ...

        /* Create or initialize the transformation by using
         * the configuration.
         */
        ...
    }
}
```

The following sample code shows how to implement the `VDSTransform` interface to shard events:

```
package ...;

import java.io.IOException;
import java.nio.ByteBuffer;
import com.informatica.vds.api.VDSConfiguration;
import com.informatica.vds.api.VDSEvent;
import com.informatica.vds.api.VDSEventList;
import com.informatica.vds.api.VDSTransform;

public class CustomTransform implements VDSTransform {

                @Override
                    public void close() throws IOException {
        /* Clean up. */
        ...
                    }

        /* Apply a transformation to the data. */
                @Override
           public void apply(VDSEvent vdsevent, VDSEventList vdsevents) throws Exception
{
              System.out.println("Applying transform...");
              ByteBuffer byteBuffer = vdsevent.getBuffer();
              byte[] event = new byte[vdsevent.getBufferLen()];
              byteBuffer.get(event);
              ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
              outputStream.write(event);
              outputStream.write(txtField.getBytes());
              System.out.println("Transformed data " + outputStream.toString());

              Map<String, String> map = new HashMap();

              String str = new String(event);

              if (str.contains("secure")) {
                  map.put("transportTopic", "SecureTopic");
              } else {
                  map.put("transportTopic", "NonsecureTopic");
              }

              vdsevents.addEvent(outputStream.toByteArray(),
outputStream.toByteArray().length, map);
```

```
                                }
                    @Override
                        public void open(VDSConfiguration vdsconfiguration) throws Exception
{

        /* The internal name of the key. */
        String exprVal = "ExpressionFromUI";

        /* The default value of the key. */
        String exprDefault = "DefaultValue"

        /* Retrieve the value of the key ExpressionFromUI. */
        String Expr = vdsconfiguration.optString(exprVal, exprDefault);

        /* Similarly, retrieve other keys. */
        ...

        /* Create or initialize the transformation by using
         * the configuration.
         */
        ...
        }
}
```

Implementing the VDSPluginStatistics Interface

Implement the `VDSPluginStatistics` interface to create statistics for entities.

The following sample code shows how to implement the `VDSPluginStatistics` interface:

```
package ...;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

import org.json.JSONArray;
import org.json.JSONObject;

import com.informatica.vds.api.VDSConfiguration;
import com.informatica.vds.api.VDSEventList;
import com.informatica.vds.api.VDSPluginStatistics;
import com.informatica.vds.api.VDSSource;

public class CustomSource implements VDSSource, VDSPluginStatistics {

    private String txtField;
    private BlockingQueue queue;
    private Map<Short, Long> statValues = new HashMap<Short, Long>();
    private long count;
    private long statistic1;
    private long statistic2;

    @Override
    public void close() throws IOException {
        System.out.println("Closing custom source...");
    }

    @Override
    public void open(VDSConfiguration arg0) throws Exception {
        System.out.println("Opening custom source...");
        txtField = arg0.getString("newField");
        String pluginStats = arg0.getString("statistic");
        JSONArray pluginStatsJsonArray = new
JSONObject(pluginStats).getJSONArray("statistic");

        for (int i = 0; i < pluginStatsJsonArray.length(); i++) {
```

```
            JSONObject stat = pluginStatsJsonArray.getJSONObject(i);
            short pluginStatKey = Short.parseShort(stat.getString("id"));
            statValues.put(new Short(pluginStatKey), (long) 0);
        }

        queue = new LinkedBlockingQueue(100);

        for (int i = 0; i < 100; i++) {
            queue.offer(txtField + i);
        }
    }

    @Override
    public void read(VDSEventList arg0) throws Exception {
        System.out.println("Reading source...");
        Thread.sleep(10);
        String message = (String) queue.poll();
        if (message == null) {
            return;
        }
        count++;
        arg0.addEvent(message.getBytes(), message.getBytes().length);

        if (count % 2 == 0) {
            statValues.put((short)2, ++statistic1);
        }
        if (count % 3 == 0) {
            statValues.put((short)3, ++statistic2);
        }

        System.out.println("Data " + message.getBytes() + " " +
message.getBytes().length);
    }

    @Override
    public long[] getStatistics(short[] arg0) {
        long[] statistics = new long[arg0.length];
        for (int i = 0; i < arg0.length; i++) {
            statistics[i] = statValues.get(arg0[i]);
        }
        return statistics;
    }
        @Override
        public void setRetryPolicyHandler(IPluginRetryPolicy arg0) {
        // TODO Auto-generated method stub
        }
}
```

Creating the EDS Plug-in JAR File

Use the `jar` command to create a JAR file that contains your implementation of the EDS interface.

Adding Dependent Libraries

Add the Java class libraries and native libraries on which your plug-in depends.

1. Navigate to the directory that contains the plug-in file and the XML file.
2. Create a directory with the name `lib`.
3. In the `lib` directory, add the Java classes on which your plug-in depends.
4. Create a directory with the name `native`.

5. In the `native` directory, add any native libraries that your implementation requires.

   Note: To include all dependencies that your implementation requires, you can use the Apache Maven Shade Plugin. For more information about the Apache Maven Shade Plugin, see http://maven.apache.org/plugins/maven-shade-plugin/.

Creating a Package for the Custom Entity Type

Create a ZIP file that contains the EDS plug-in XML document, the EDS plug-in JAR file, and the `lib` and `native` directories.

Using a Custom Entity Type

To use a custom entity type, register the EDS plug-in with the Administrator Daemon. To stop using a custom entity type, delete the entity from the data flow and unregister the EDS plug-in with the Administrator Daemon.

To register or unregister a plug-in, perform the following tasks:

- Stop the Administrator Daemon.
- Register or unregister the EDS plug-in.
- Start the Administrator Daemon.

Stopping the Administrator Daemon

To stop the Administrator Daemon, you use a script that the EDS installer saves to the `<EDS installation directory>/bin/` directory.

Stopping the Administrator Daemon on Linux

On a Linux machine, use the command-line interface to stop the Administrator Daemon.

1. At the command prompt, navigate to the following directory: `<EDS installation directory>\bin\`
2. To stop the Administrator Daemon, run the following command:

   ./admind.sh stop admind

Stopping the Administrator Daemon on Windows

Use the Services window in the Control Panel to stop the Administrator Daemon service.

1. Open the Windows Control Panel.
2. Select **Administrative Tools**.
3. Select **Services**.
4. Right-click the **ADMIND** service.
5. To stop the service, click **Stop**.

Registering or Unregistering an EDS Plug-in

Save the EDS plug-in to any directory on the Administrator Daemon host. Stop the Administrator Daemon. To register the EDS plug-in, use a script that the installer creates in the Administrator Daemon installation directory.

Ensure that Apache ZooKeeper is running when you register the plug-in as ZooKeeper contains the database details required to register the plug-in.

Registering or Unregistering an EDS Plug-in on Linux

To register or unregister an EDS plug-in on a Linux machine, use the plugin.sh script.

1. At the command prompt, navigate to the following directory:

   `<EDS installation directory>/admind/bin`
2. To register the plug-in with the Administrator Daemon, run the following command:

   `./plugin.sh register <zipFile>`
   where `<zipFile>` is the full path to the plug-in that you want to register.
3. To unregister the plug-in with the Administrator Daemon, run the following command:

   `./plugin.sh unregister <pluginID>`
   where `<pluginID>` is the ID of the plug-in that you want to unregister.

Registering or Unregistering a EDS Plugin on Windows

To register or unregister a EDS plug-in on a Windows machine, use the plugin.bat script. Run the script from the Windows command line.

1. At the Windows command prompt, navigate to the following folder: `<EDS installation directory>\admind\bin`
2. To register the EDS plug-in with the Administrator Daemon, run the following command:

   `plugin.bat register <zipFile>`
   where `<zipFile>` is the full path to the EDS plug-in that you want to register.
3. To unregister the EDS plug-in with the Administrator Daemon, run the following command:

   `plugin.bat unregister <pluginID>`
   where `<pluginID>` is the ID of the EDS plug-in that you want to unregister.

Starting the Administrator Daemon

To start the Administrator Daemon, you use a script that the EDS installer saves to the `<EDS installation directory>/admind/bin` directory.

Starting the Administrator Daemon on Linux

On a Linux machine, use the command-line interface to start the Administrator Daemon.

1. At the command prompt, navigate to the following directory: `<EDS installation directory>/admind/bin`
2. To start the Administrator Daemon, run the following command: ./admind.sh start admind

Starting the Administrator Daemon on Windows

Use the Services window in the Control Panel to start the Administrator Daemon service.

1. Open the Windows Control Panel.
2. Select **Administrative Tools**.
3. Select **Services**.
4. Right-click the **ADMIND** service.
5. To start the service, click **Start**.

Versioning a Custom Entity Type

To make changes to a custom entity type that you added to Edge Data Streaming, upgrade the entity type with a newer version.

To upgrade or downgrade a custom entity type, perform the following tasks:

- Edit the implementation of the custom entity.
- Increment or decrease the version number in the Edge Data Streaming plug-in XML file.
- Stop the Administrator Daemon.
- Optionally, prepare the custom entity if you make changes to the Edge Data Streaming plug-in XML file. In the CSV file, add the rows for the newly added fields.
- Upgrade the entity type.
- Start the Administrator Daemon. When you start the Administrator Daemon, the Administrator Daemon deploys the data flows that were undeployed during the upgrade.
- Verify if the changes to the custom entity appear in the Administrator tool.

For information about undeploying and deploying the data flow and stopping and starting the Administrator Daemon see *Informatica Edge Data Streaming User Guide*.

Editing the Custom Entity Implementation

Edit the EDS plug-in XML file and the EDS plug-in JAR file. Increment or decrease the version number in the EDS plug-in XML file.

Create a ZIP file that contains the modified EDS plug-in JAR file, the EDS plug-in XML file with the incremented version number, and any libraries on which the plug-in depends.

Incrementing the Version Number

To upgrade the custom entity type, increment the version number in the EDS plug-in XML file.

Open the EDS plug-in XML file that the custom entity type uses, increment the integer value in the `version` XML element, and then save and close the file.

Preparing a Custom Entity

Prepare the custom entity if you make changes to the EDS plug-in XML file.

1. At the command prompt, navigate to the following directory:

   `<EDS installation directory>/admind/bin`

2. Run the following command to prepare the custom entity:

   On Linux:

   `plugin.sh prepare <pluginid> <directory where the CSV will be written>`

   On Windows:

   `plugin.bat prepare <pluginid> <directory where the CSV will be written>`

   The CSV file is created.

Editing the CSV File

Edit the CSV file that is created when you run the prepare command. Add the new values that you define for the fields you changed or added in the EDS plug-in XML file.

1. Navigate to the directory where you generated the CSV file.
2. Use a text editor to edit the CSV file.

   Note: Do not edit the `ENTITY_DESIGN_ID` and `_VDS_ENTITY_NAME` fields.

3. Add, change or delete columns and the corresponding values in the CSV file. Enclose the values in double quotes.

   Note: You must specify the values for fields that you add even if you specify the default values while describing the fields in the Plug-in XML file.

4. Save and close the CSV file.

Upgrading a Custom Entity Type

To upgrade a custom entity type, replace the existing ZIP file with a ZIP file that contains the modified implementation of the entity type.

1. Log in to the host on which you installed the Administrator Daemon and stop the Administrator Daemon.
2. Back up the existing version of the ZIP file, and then replace it with the new version.
3. To upgrade the custom entity only if you have made changes to the EDS plug-in JAR file, run the following command:

   On Linux:

   `plugin.sh upgrade <ZIP file>`

   On Windows:

   `plugin.bat upgrade <ZIP file>`

   where `ZIP file` is the full path to the EDS plug-in ZIP file

4. To upgrade the custom entity if you have made changes to the EDS plug-in XML file, run the following command:

   On Linux:

   `plugin.sh upgrade <ZIP file> <CSV file>`

   On Windows:

```
plugin.bat upgrade <ZIP file> <CSV file>
```

where `ZIP file` is the full path to the EDS plug-in ZIP file and `CSV file` is the full path to the CSV file.

5. Start the Administrator Daemon. When you start the Administrator Daemon, the Administrator Daemon deploys the data flows that were undeployed during registration or unregistration.
6. Verify if the changes to the custom entity appear in the Administrator tool.

   Note: To downgrade a custom entity, replace the existing ZIP file with the earlier ZIP file version.

Related Topics:

- "Creating a Package for the Custom Entity Type" on page 30
- "Creating the EDS Plug-in JAR File" on page 29
- "Stopping the Administrator Daemon" on page 30

Custom Source Service Type Example

This example shows how you can create a custom source service type and use it in your data flow.

Step 1: Create the Edge Data Streaming Plug-in XML Document

Create an XML document called `vdsplugin.xml`. In the XML document, describe the configuration of the custom entity type.

You can use the following sample configuration in the XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:vdsPlugin xmlns:tns="http://www.informatica.com/VdsPlugin"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.informatica.com/VdsPlugin ../XSD/eds_plugin.xsd ">
    <tns:id>SimpleSrc</tns:id>
    <tns:displayName>Simple Source</tns:displayName>
    <tns:version>1.0</tns:version>
    <tns:type>SOURCE</tns:type>
    <tns:configuration>
        <tns:fields>
            <tns:field>
        <tns:textControl>
            <tns:name>message</tns:name>
            <tns:displayName>Message</tns:displayName>
            <tns:description>Message to be sent</tns:description>
            <tns:mandatory>true</tns:mandatory>
            <tns:stringTextField>
            <tns:secure/>
            </tns:stringTextField>
        </tns:textControl>
      </tns:field>
        </tns:fields>
    </tns:configuration>
    <tns:runTime>
        <tns:pluginJar>SimpleSource.jar</tns:pluginJar>
        <tns:pluginClass>com.informatica.vds.plugin.custom.SimpleSource</tns:pluginClass>
    </tns:runTime>
  <tns:helpKey></tns:helpKey>
</tns:vdsPlugin>
```

Step 2: Create the Java Source File

Create a Java source file called `SimpleSource.java`.

You can use the following sample Java source file:

```
package com.informatica.vds.plugin.custom;

import java.io.IOException;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

import com.informatica.vds.api.VDSConfiguration;
import com.informatica.vds.api.VDSEventList;
import com.informatica.vds.api.VDSSource;

public class SimpleSource implements VDSSource {

    private String message;
    private BlockingQueue queue;
    private long count;

    @Override
    public void close() throws IOException {
        System.out.println("Closing custom source...");
    }

    @Override
    public void open(VDSConfiguration arg0) throws Exception {
        System.out.println("Opening custom source...");
        message = arg0.getString("message");

        queue = new LinkedBlockingQueue(100);

        for (int i = 0; i < 100; i++) {
            queue.offer(message + i);
        }
    }

    @Override
    public void read(VDSEventList arg0) throws Exception {
        System.out.println("Reading source...");
        Thread.sleep(100);
        String message = (String) queue.poll();
        if (message == null) {
            return;
        }
        arg0.addEvent(message.getBytes(), message.getBytes().length);
    }
}
```

Step 3: Create the EDS Plug-in JAR File

Use the jar command to create a `SimpleSource.jar` JAR file that contains your implementation of the EDS interface.

Add the `vds-api-<version>.jar` to the Java build path so that the simple source can be compiled.

Step 4: Create a Package for the Custom Entity Type

Create a folder named `simplesource`. Place the `SimpleSource.jar` and `vdsplugin.xml` files in this folder and create a ZIP file called `simplesource.zip`.

Step 5: Register the EDS Plug-in

Save the `simplesource.zip` plug-in to any directory on the Administrator Daemon host. Stop the Administrator Daemon. To register the EDS plug-in, use a script that the installer creates in the Administrator Daemon installation directory.

Navigate to the following directory:

```
<EDS installation directory>/admind/bin
```

To register the `simplesource.zip` plug-in with the Administrator Daemon on a Linux machine, run the following command:

```
./plugin.sh register <full path to simplesource.zip>
```

To register the EDS plug-in with the Administrator Daemon on a Windows machine, run the following command:

```
plugin.bat register <full path to simplesource.zip>
```

After you register the `simplesource.zip` plug-in, start the Administrator Daemon.

Step 6: Use the Plug-in in a Data Flow

To use the `Simple Source` plug-in a data flow, log in to the Administrator tool.

Navigate to the **Edge Data Streaming** tab.

Verify that you can view **Simple Source** listed under the sources in the **Entity Types** panel.

You can use the **Simple Source** source service in your data flows.

Troubleshooting Custom Entity Types

The custom entity does not appear in the Administrator tool:

This error occurs if the plug-in is not registered correctly. Check the `<EDS Installation Directory>/admind/logs/register.log` file for errors and correct them.

I get an error when I register a custom entity.

This error occurs if you are using the **Send to** > **Compressed (zipped) folder** option in Windows to create the custom entity ZIP file. Use a different ZIP utility to create the custom entity ZIP file. For example, you can use the Windows WinZip utility.

I get an error when I upgrade a custom entity.

This error can occur if ZooKeeper is down when you upgrade the custom entity. A ZooKeeper connectivity exception occurs in the `<EDS Installation Directory>/admind/logs/upgrade.log` file.

Navigate to the following directory:

```
<EDS installation directory>/admind/bin
```

To verify the status, run the following command:

```
./admind.sh status
```

If you have installed a version of ZooKeeper that is not part of the EDS installation package, navigate to the following directory:

```
<EDS installation directory>/zookeeper/bin
```

To verify the status, run the following command:

```
./zkServer.sh status
```

If ZooKeeper is not running, use the following command to start it:

```
./zkServer.sh start
```

The custom entity that I created does not behave as expected.

To identify the cause for the unexpected behavior, you can attach a debugger to the EDS Node.

To attach a debugger, perform the following steps:

1. Navigate to the following directory:
   `<EDS installation directory>/node/bin`
2. Edit the `node.bat` file on Windows and edit the `node.sh` file on Linux.
   Include the following line in the `JAVA_OPTS` configuration:

   ```
   -Xdebug;-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9699
   ```

   For example, `SET "JAVA_OPTS=-Xms256m;-Xmx2G;-XX:+UseCompressedOops;-XX:+UseConcMarkSweepGC;-XX:CMSInitiatingOccupancyFraction=70;-XX:MaxDirectMemorySize=512m;-Dio.netty.leakDetectionLevel=disabled;-DINFA_HOME=%PRODUCT_HOME%;-Dnodename=%NODENAME%;-Dhostname=%HOSTNAME%;-Dlogdir=%LOGS_DIR%;-Dlog4j.configuration=%PRODUCT_HOME%/config/log4j.properties` **`-Xdebug;-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9699"`**
3. Save and close the file.

On Windows, additionally perform the following steps:

1. Run the following command to uninstall the EDS Node:
   Node.bat uninstall <hostname>
2. Run the following command to install the EDS Node:
   Node.bat install <hostname>

   You can attach the EDS Node to a remote debugger with the port 9699.

When I run the Maven install command to create a custom entity using the Maven archetypes, I get the following warnings in the Eclipse console:

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

These warnings occur if you are using the Eclipse version Kepler or earlier versions. Upgrade the Eclipse version to Luna or a newer version.

CHAPTER 3

Custom Entities from Maven Archetypes

This chapter includes the following topics:

Custom Entities from Maven Archetypes Overview

You can use the Maven archetypes to create entities, such as source services, target services, and transformations. The Maven archetypes are packaged with the EDS APIs. You can use these archetypes to create custom entity types in an Eclipse environment.

An Apache Maven archetype is a model or pattern that contains the prototypes of the entities that you want to create. You can use the archetype plug-in to create a Maven project from a template in Eclipse. Eclipse is a multi-language software development environment that includes a base workspace and an extensible plug-in system for customizing the environment.

The archetype information is stored in catalogs, which are XML files. The archetype plug-in contains an internal catalog that is used by default. You can also use catalogs from remote repositories.

EDS includes the following Maven archetypes:

- `vds-template-source-archetype`. The archetype for the source service.
- `vds-template-target-archetype`. The archetype for the target service.
- `vds-template-transform-archetype`. The archetype for the transformation.

Create an Entity Based on an Archetype in the Internal Catalog

The archetype plug-in can use an internal catalog.

To create entities based on archetypes in the internal catalog, perform the following steps:

1. Verify the prerequisites.
2. Copy the EDS API JAR file and the archetypes from the `EDS-api-reference.zip` ZIP file and to the local Maven repository.
3. Copy the `archetype-catalog.xml` file to the root of the Maven directory.
4. Add the local archetype catalog to Eclipse.
5. Create a Maven project with the archetypes you copy.
6. Install the project.

Step 1: Verify the Prerequisites

Before you begin, perform the following tasks:

- Verify that the Eclipse environment uses JDK 1.7.
- Install Apache Maven 3.x or a later version.
- Verify that you have extracted the contents of `EDS-api-reference.zip` file.

Step 2: Copy the API JAR File and the Archetypes to the Local Repository

Copy the extracted `vds-api.jar` and the archetypes to the local Maven repository. The local Maven repository stores the plugin jar files and other files that are downloaded by Maven.

The archetypes are located in the following directory:

`EDS-api-reference\archetypes`

1. Set the `M2_HOME` environment variable to the Maven installation directory.
2. Set the `PATH` to the `M2_HOME/bin` directory.
3. Verify that the `JAVA_HOME` is set to the directory where the JDK is installed.
4. Open a command prompt in Windows or a terminal in Linux.
5. Navigate to the folder where you extracted the files from the `EDS-api-reference.zip` file.
6. Navigate to the `scripts` folder.
7. To copy the API JAR file and the archetypes to the local repository, run the following command:
   - install-eds-api.bat. Run this command on Windows.
   - /install-eds-api.sh. Run this command on Linux.

Step 3: Copy the Archetype Descriptor to the Home Directory

To copy the archetype descriptor file to the home directory, perform the following steps:

1. Go to the directory where you extracted the EDS API package.

2. Navigate to the `catalog` folder.
3. Copy the `archetype-catalog.xml` file to the `<USER_HOME>\.m2` directory.
   The following example shows a Windows path:

   ```
   C:\Users\<user name>\.m2
   ```

   The following example shows a Linux path:

   ```
   /home/<user ID>/.m2/
   ```

Note: If the `archetype-catalog.xml` file already exists, copy the code in the `<archetypes>` element of the `archetype-catalog.xml` file in the EDS API package to the existing file.

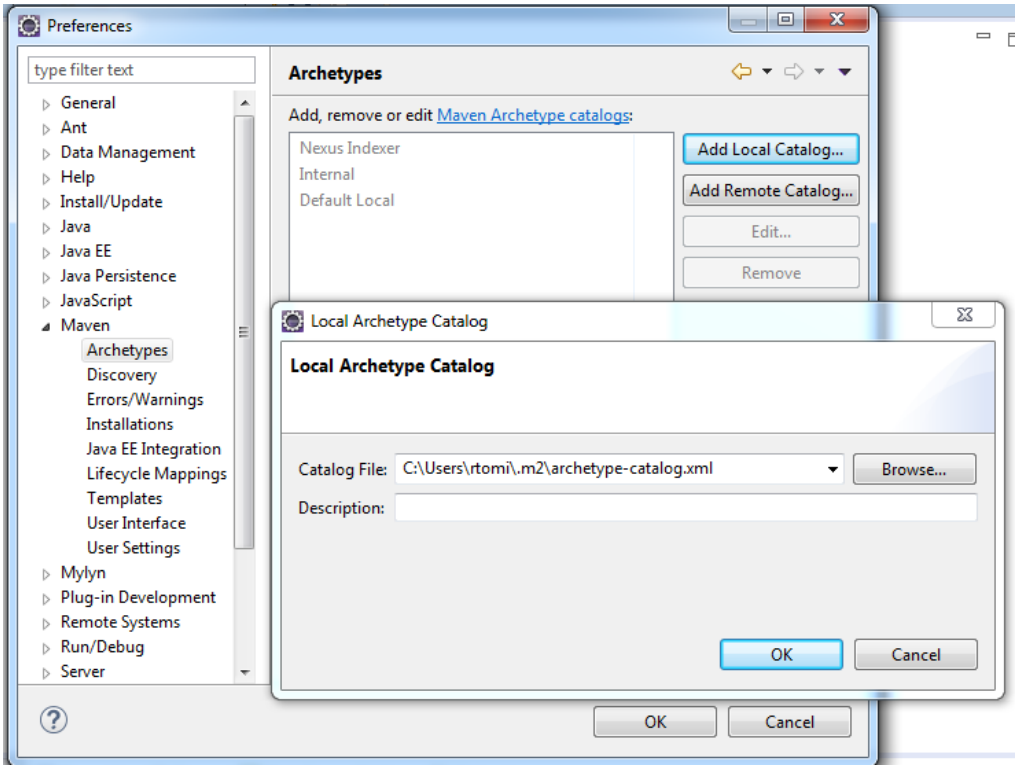
Step 4: Add the Local Archetype Descriptor File

To use the local archetype descriptor, add it to Eclipse.

1. Start Eclipse.
2. Go to **Window** > **Preferences** > **Maven** > **Archetypes**
3. Click **Add Local Catalog**.

   The **Local Archetype Catalog** window opens.
4. Browse to the location of the `archetype-catalog.xml` file and click **Open**.

   The following image shows the **Local Archetype Catalog** window:



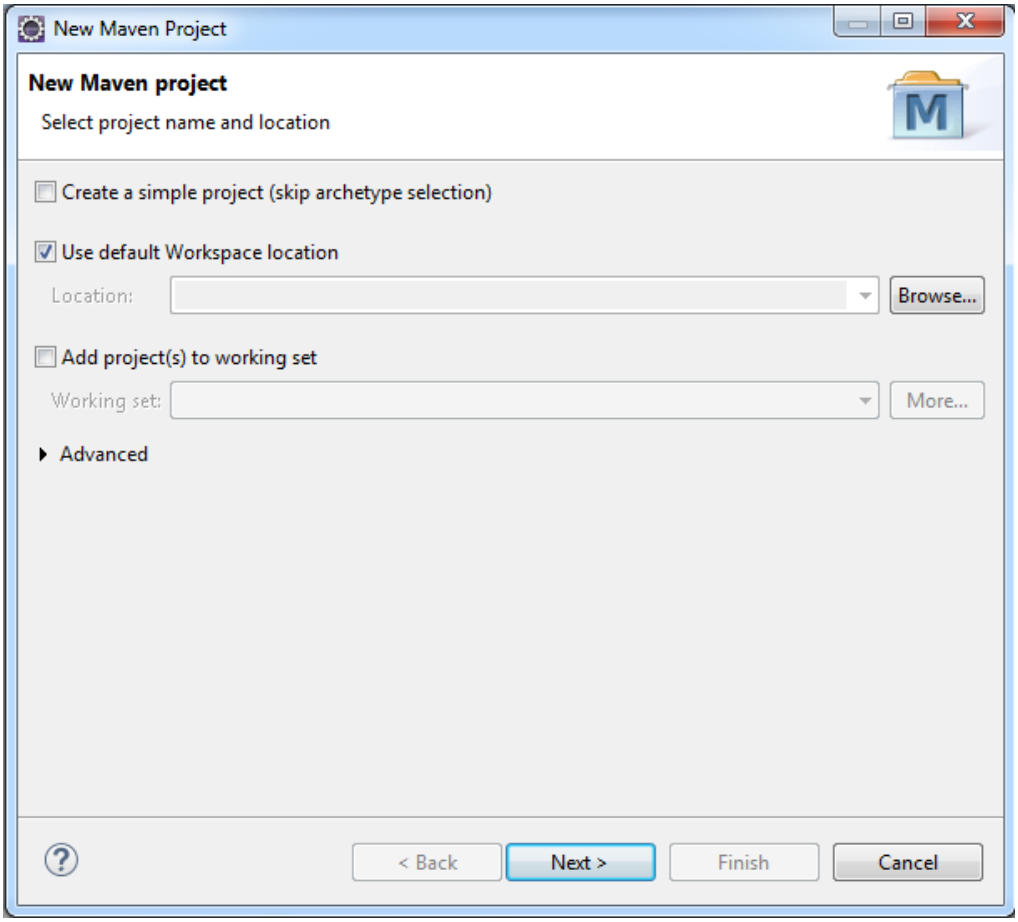
5. Click **OK**, **Apply**, and **OK** to add the catalog.

Step 5: Create a Maven Project

Create a Maven project to use the archetypes.

1. On the Eclipse Workbench window, click **File** and select **New** > **Maven Project**.

   The **Select project name and location** page of the **New Maven Project** wizard appears.
   The following image shows the **New Maven Project** wizard:



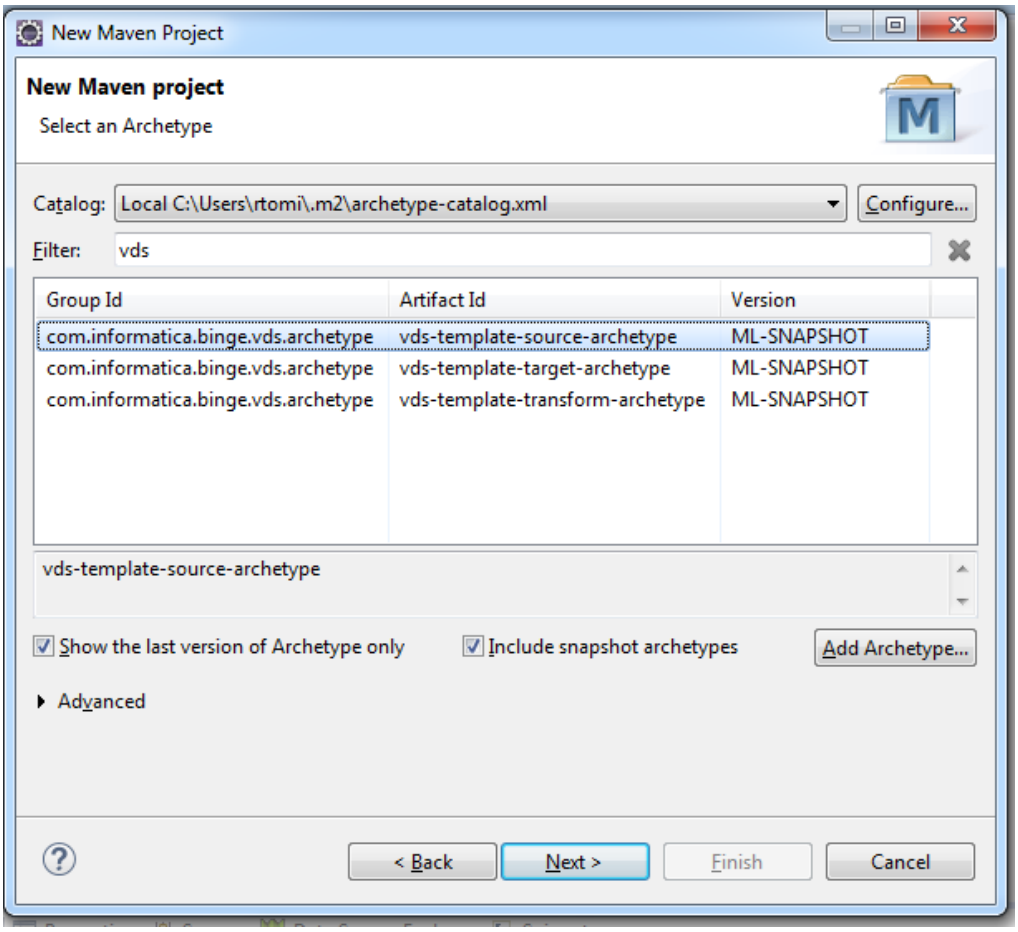
   Note: Do not select **Create a simple project (skip archetype selection)**.

2. Click **Next**.

   The **Select an Archetype** page appears.

3. Select **Local <USER_HOME>/.m2/archetype-catalog.xml** from the **Catalog** list.
4. Filter the archetypes using keyword **vds** because the Artifact ID of the EDS archetypes begin with **vds**.
5. Select the archetype from the list.

   The following image shows the **vds-template-source-archetype** archetype selection:



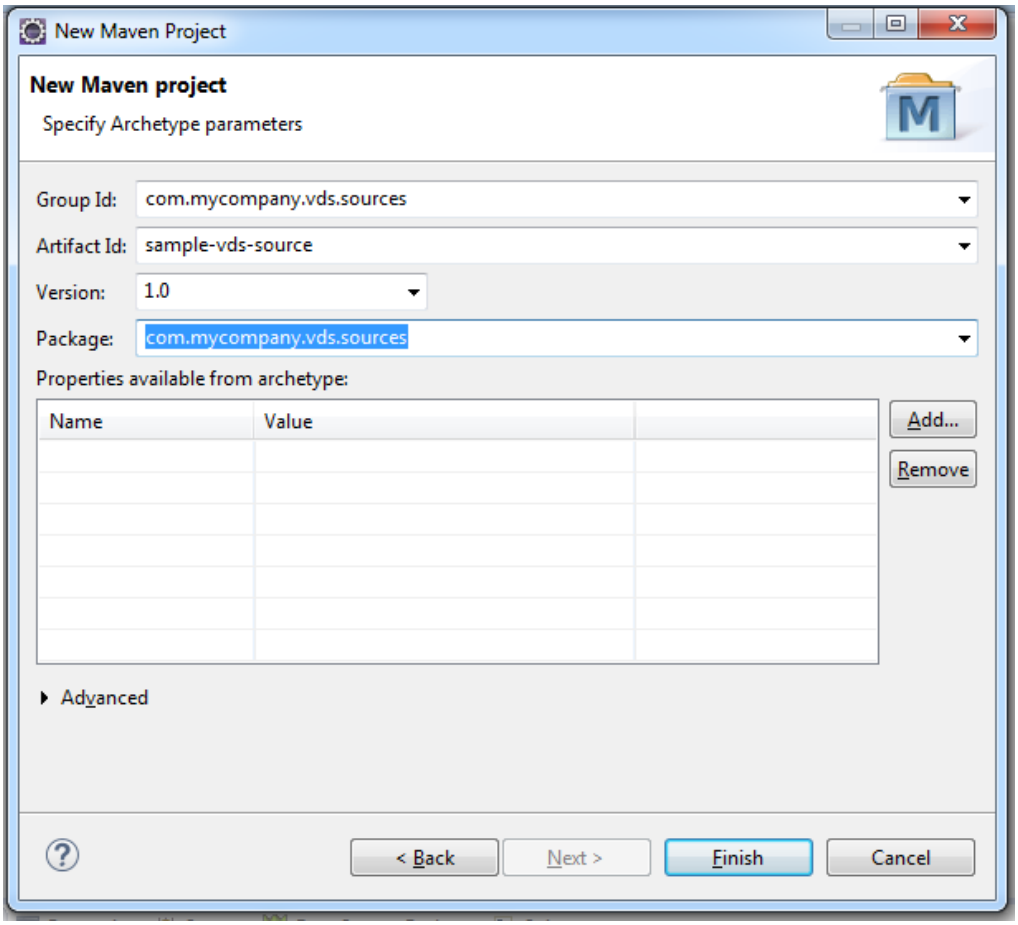
6. Click **Next**.

   The **Specify Archetype parameters** page appears.

7. Enter the following values:

Field	Value
Group Id	Group ID that identifies the project. Example: `com.mycompany.vds.sources`
Artifact Id	Name of the jar file without the version. Example: `sample-vds-source`
Version	Version number of the artifact. Example: `1.0`
Package	Name of Java package where the source service, target service, and transformation classes are created. Example: `com.mycompany.vds.sources`

The following image shows the **Archetype parameters**:

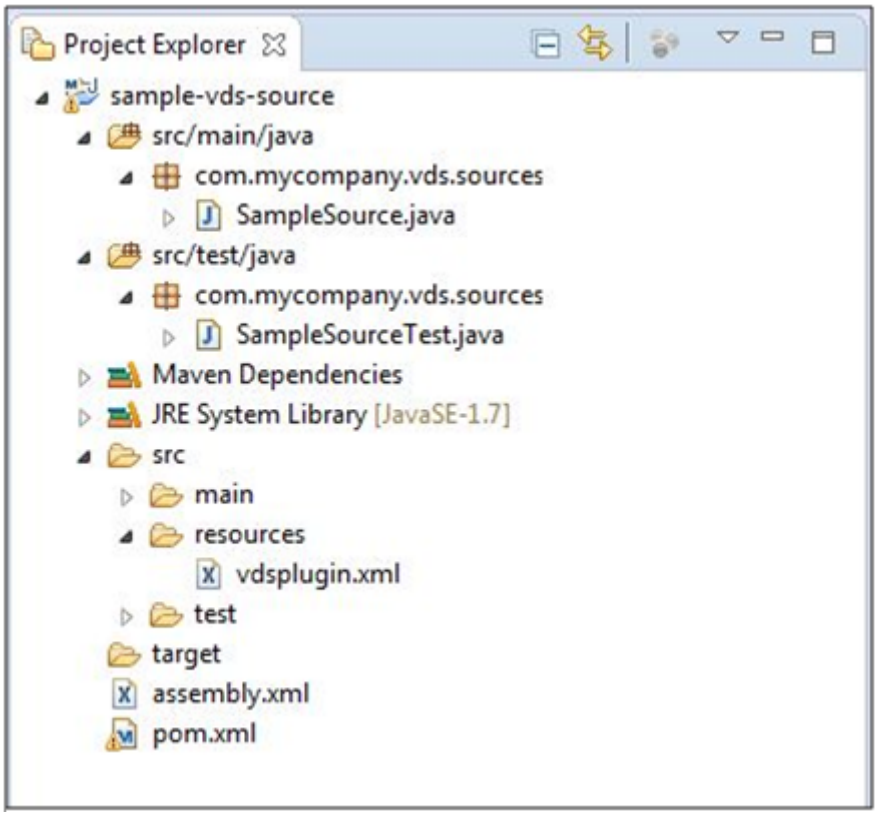


8. Click **Finish**.

   To view the progress of the background operations, click **Progress** view.

   The `sample-vds-source project` is created under your current workspace.

   The following image shows the sample project:



Step 6: Install the Maven Project

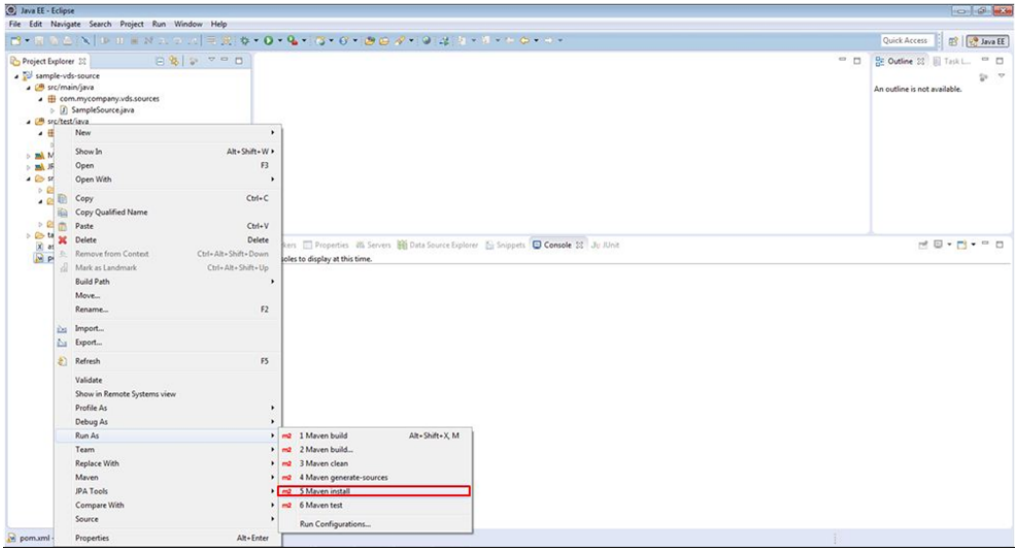
After you create the Maven project, install the project to create the custom plug-in.

Before you install the Maven project, remove files generated in the project directory when you created a project previously by running the `Maven Clean` command. In the Project Explorer, right click the project. Select **Run As** > **Maven clean**.

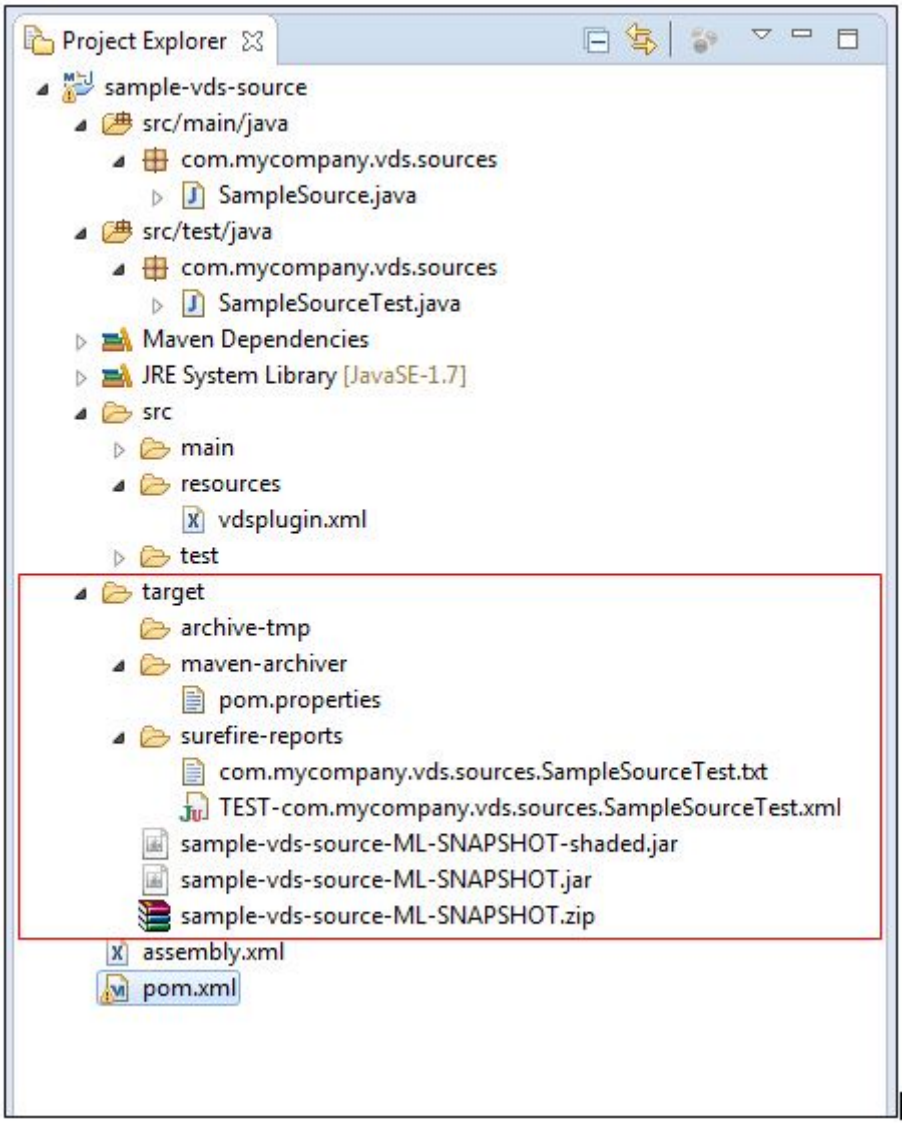
1. Select the project in the project explorer.

2. Right-click and select **Run As** > **Maven install**.

   The following image shows the `Maven install` command:



   The following image shows the target directory after the Maven command has run successfully:



Create an Entity Based on an Archetype in a Remote Catalog

The archetype plug-in can use catalogs that are located in remote locations.

To create entities from catalogs in remote locations, perform the following steps:

1. Verify the prerequisites.
2. Add the remote archetype descriptor file in Eclipse.
3. Create a Maven project with the archetypes.
4. Install the project.

Step 1: Verify the Prerequisites

Before you begin, perform the following tasks:

- Verify that the Eclipse environment uses JDK 1.7.
- Install Apache Maven 3.x or a higher version.
- Copy the archetypes to the remote Maven repository.
- Verify that the `settings.xml` file has the path to the remote catalog.

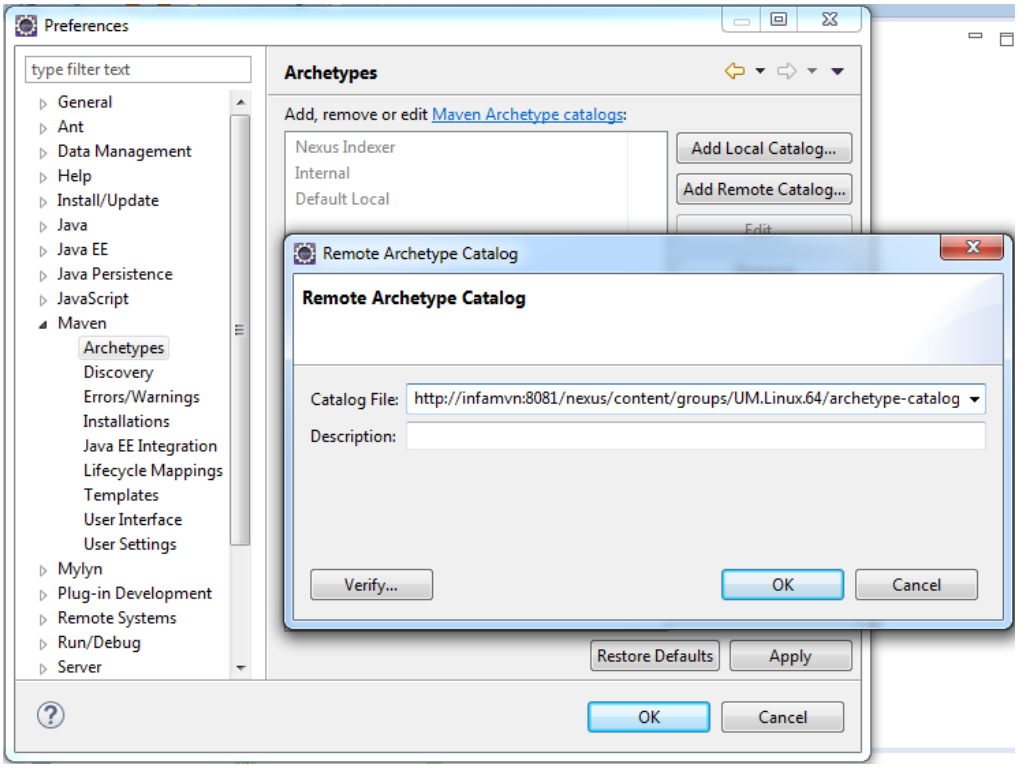
Step 2: Add the Remote Archetype Descriptor File

To use the remote archetype descriptor, add it in Eclipse.

1. Open Eclipse.
2. Go to **Window** > **Preferences** > **Maven** > **Archetypes**
3. Select **Add Remote Catalog**.

   The **Remote Archetype Catalog** dialog box opens.
4. Specify the location of the `archetype-catalog.xml` file.

   The following image shows the **Remote Archetype Catalog** dialog box:



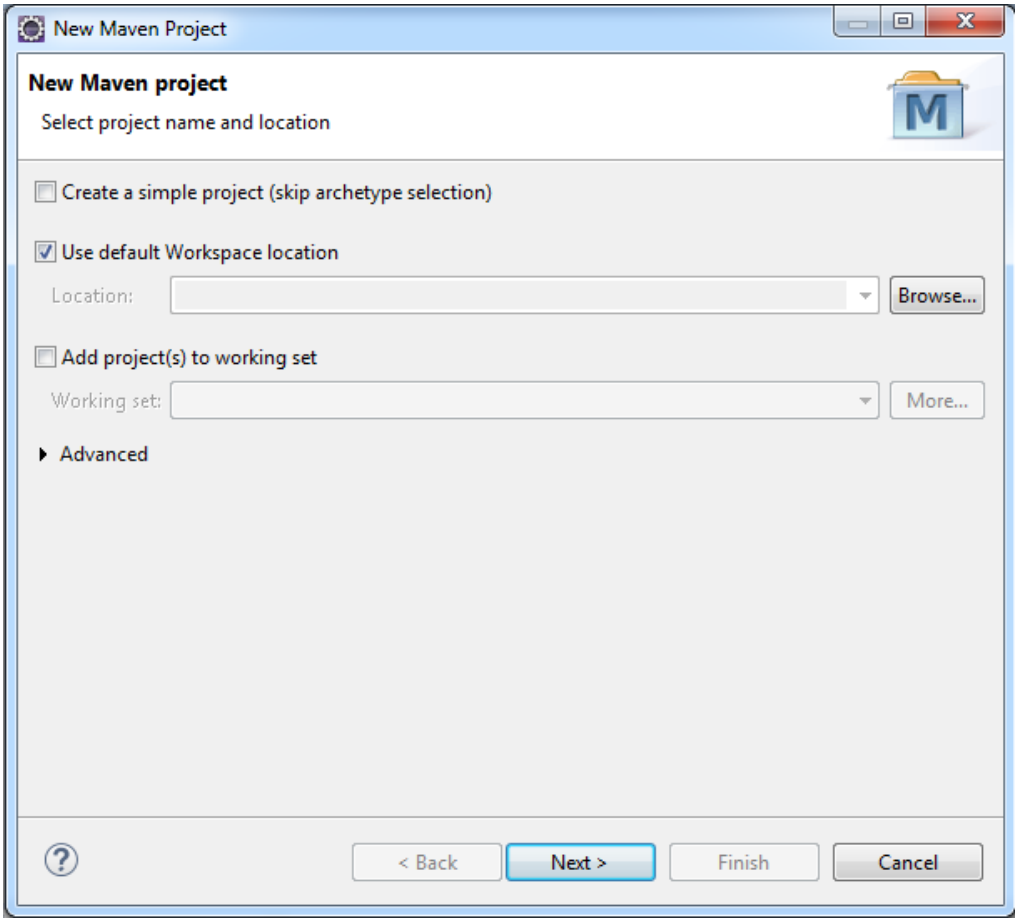
5. Click **OK**, **Apply**, and **OK** to add the catalog.

Step 3: Create a Maven Project

Create a Maven project to use the archetypes.

1. In Eclipse, click **File** and select **New** > **Maven Project**.

   The **Select project name and location** page of the **New Maven Project** wizard appears. The following image shows the **New Maven Project** wizard:



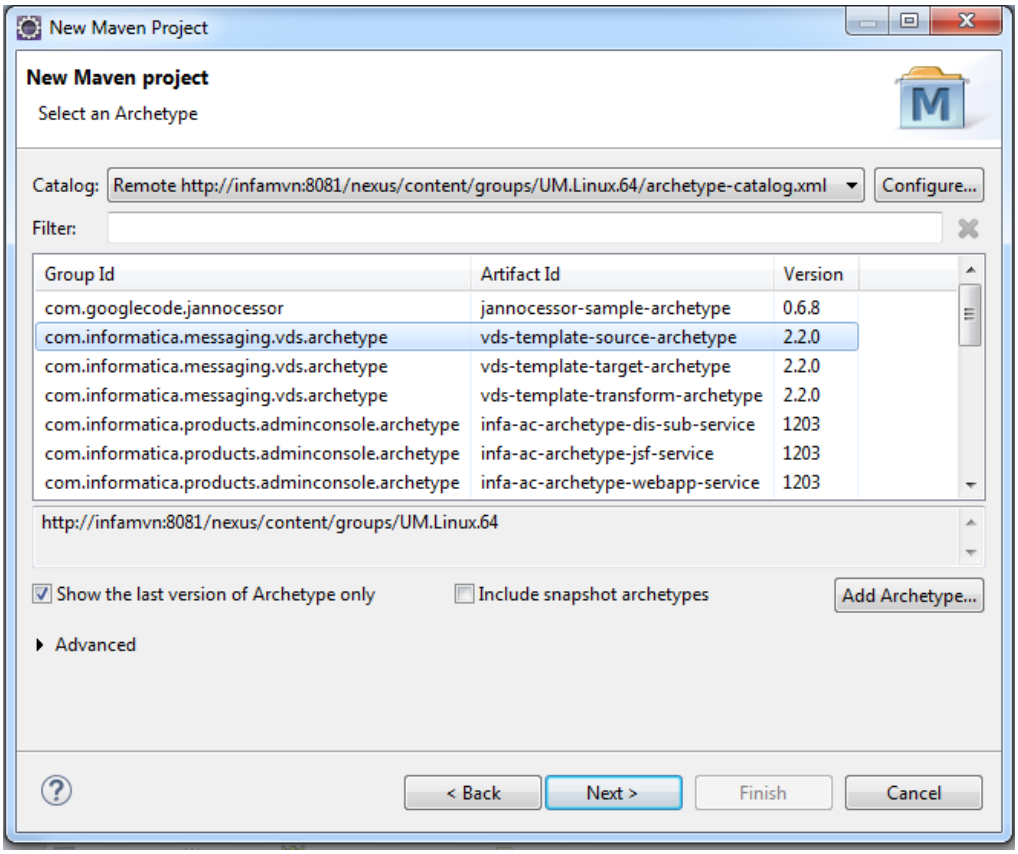
   Note: Do not select **Create a simple project (skip archetype selection)**.

2. Click **Next**.

   The **Select an Archetype** page appears.

3. Select the remote catalog from the list.
4. Filter the archetypes using keyword **vds**.
5. Select the archetype from the list.

   The following image shows the archetype catalog selection:



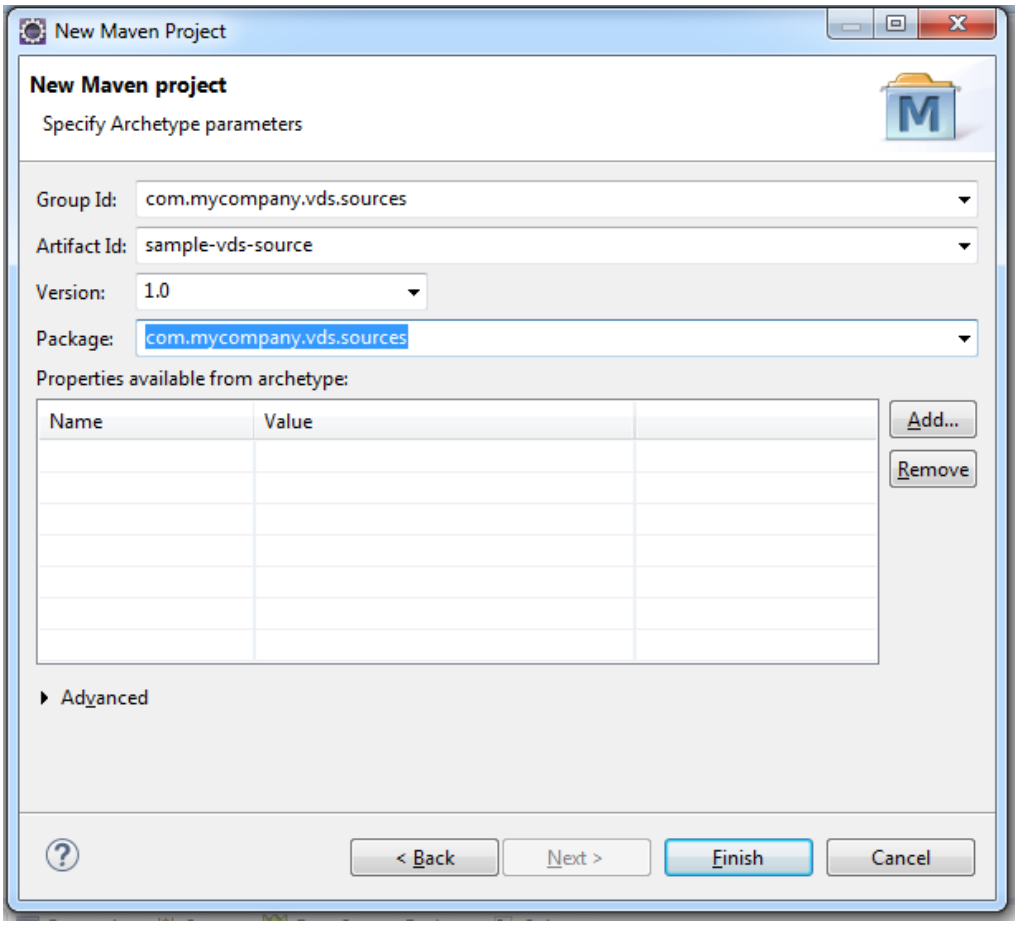
6. Click **Next**.

   The **Specify Archetype parameters** page appears.

7. Enter the following values:

Field	Value
Group Id	Group ID that identifies the project. Example: `com.mycompany.vds.sources`
Artifact Id	Name of the jar file without the version. Example: `sample-vds-source`
Version	Version number of the artifact. Example: `1.0`
Package	Name of Java package under which the source service, target service, and transformation classes are created. Example: `com.mycompany.vds.sources`

The following image shows the **Archetype parameters**:

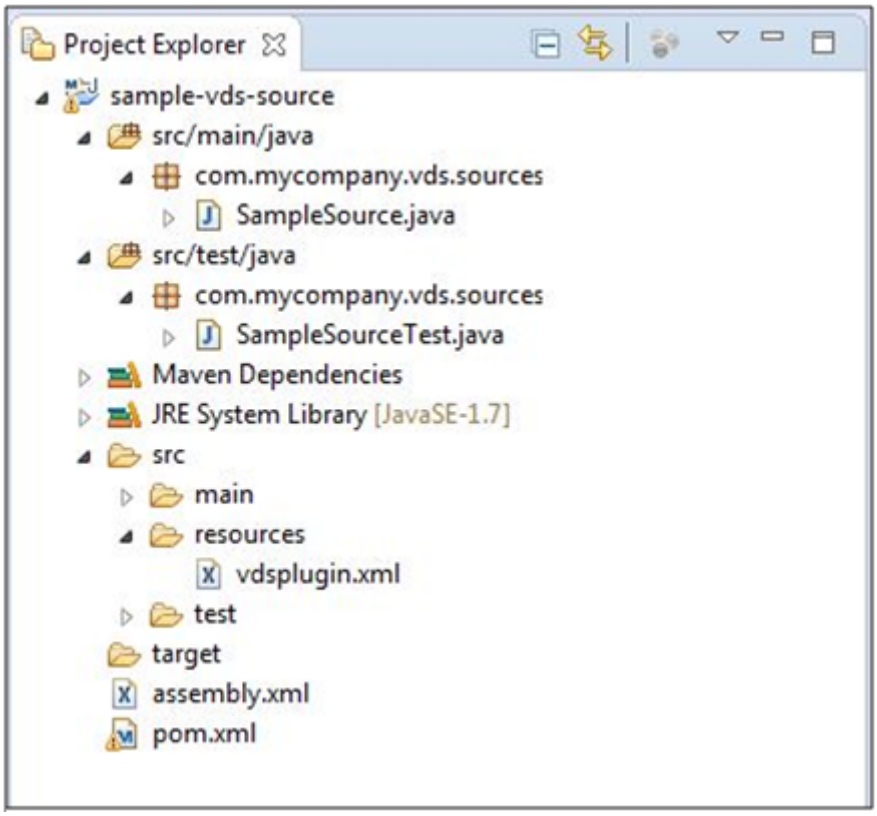


8. Click **Finish**.

   To view the progress of the background operations, click **Progress** view.

   The `sample-vds-source project` is created under your current workspace.

   The following image shows the sample project:



Step 6: Install the Maven Project

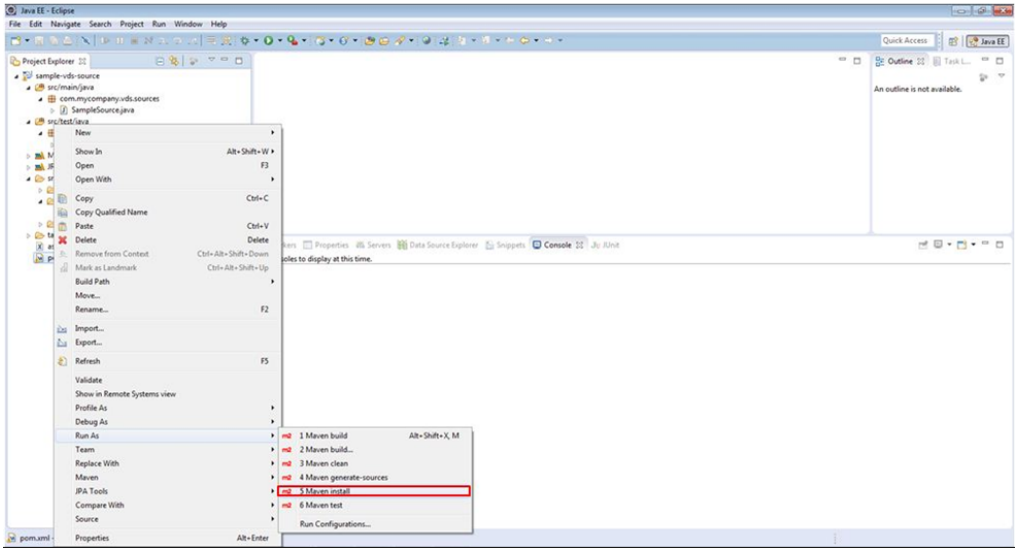
After you create the Maven project, install the project to create the custom plug-in.

Before you install the Maven project, remove files generated in the project directory when you created a project previously by running the `Maven Clean` command. In the Project Explorer, right click the project. Select **Run As** > **Maven clean**.

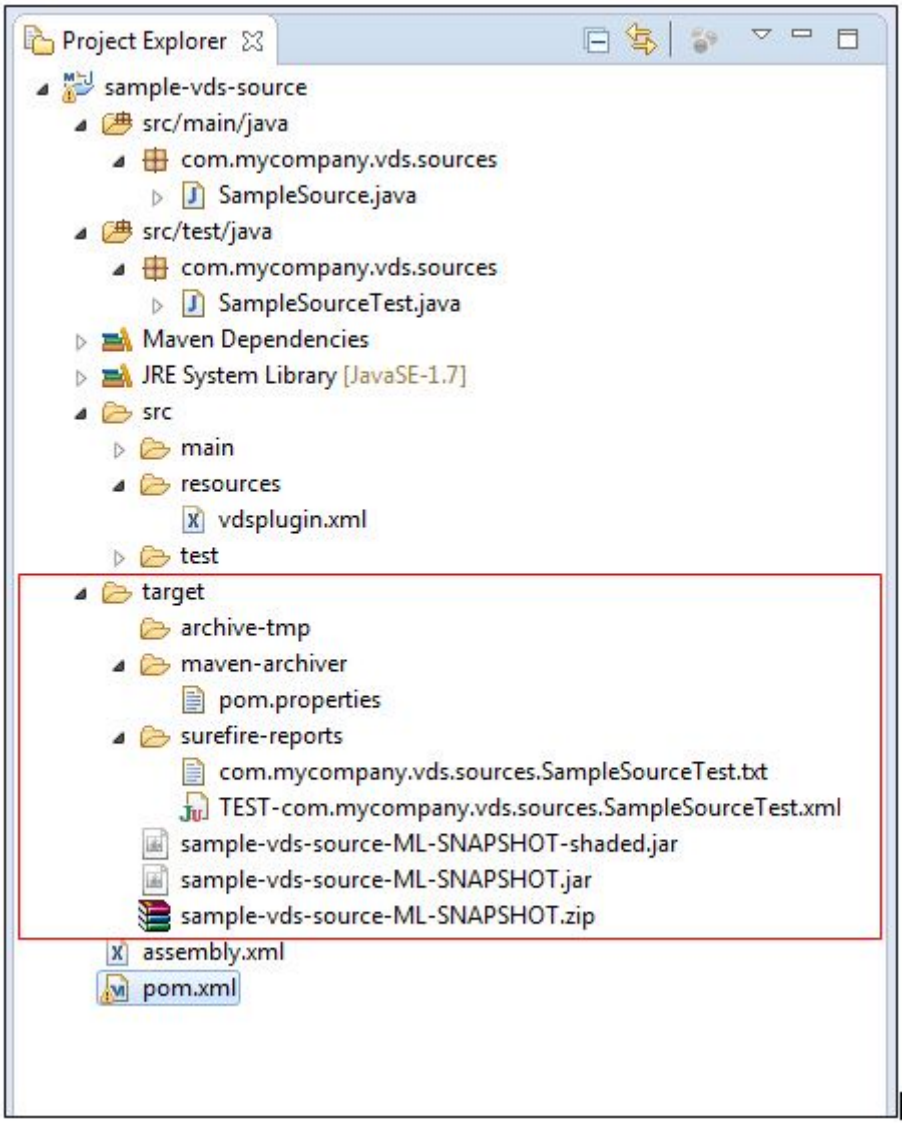
1. Select the project in the project explorer.

2. Right-click and select **Run As** > **Maven install**.

   The following image shows the `Maven install` command:



   The following image shows the target directory after the Maven command has run successfully:


Project Explorer
sample-vds-source
src/main/java
com.mycompany.vds.sources
SampleSource.java
src/test/java
com.mycompany.vds.sources
SampleSourceTest.java
Maven Dependencies
JRE System Library [JavaSE-1.7]
src
main
resources
vdsplugin.xml
test
target
archive-tmp
maven-archiver
pom.properties
surefire-reports
com.mycompany.vds.sources.SampleSourceTest.txt
TEST-com.mycompany.vds.sources.SampleSourceTest.xml
sample-vds-source-ML-SNAPSHOT-shaded.jar
sample-vds-source-ML-SNAPSHOT.jar
sample-vds-source-ML-SNAPSHOT.zip
assembly.xml
pom.xml

CHAPTER 4

REST APIs

This chapter includes the following topics:

REST APIs Overview

The REST API is a Representational State Transfer (REST) API.

The EDS REST API allows you to use API calls to perform the following tasks in EDS:

- Create, update, retrieve, delete, deploy, and undeploy specific and all data flows.
- Create, update, retrieve, delete, deploy, and undeploy source services and target services.
- Create, update, retrieve, and delete connections and links.
- Create, update, retrieve, and delete aggregators and transformations.
- Create, update, retrieve, and delete EDS Nodes and node groups.
- Create, update, retrieve, and delete parameters.
- Retrieve plug-ins.

You can access the REST APIs through the following base URL:

`http://<hostname>:<port>/administrator/api`

where, `hostname` is the machine name on which the Administrator tool is running and `port` is the port on which it listens.

When you use the REST API to configure a request, use the appropriate resource, method, the applicable date and time values, object ID, along with the applicable headers. Use the applicable format for the request header and body component. EDS supports the JSON formats for passing the attributes. EDS performs the requested task and returns the http response, or returns an error object and related messages.

For more information on the request and response format for specific EDS resources, see the sample JSON requests and responses.

Header and Body Configuration

A REST operation combines an HTTP method with the full URL to the resource. For a complete request, combine the REST operation with the HTTP headers and required data. A REST request has a header and a body component. You can use the JSON format to define a request.

Request Header

Use a request header to define the operating parameters or the metadata of the REST operation. The header consists of a series of field-value pairs. The API request line contains the method and the URL. Specify the header fields after the request line.

To construct a REST API request header, use the following format:

```
<METHOD> <Server URL>/api/<URI>
Content-Type: application/json
Accept: application/json
Cookie: $Version=1; JSESSIONID=<Valid_session_ID>
```

The following table includes the request components:

Request Component	Required/ Optional	Description
METHOD	Required	Method you want to use, such as GET, POST, PUT, or DELETE.
Server URL	Required	Base URL for all resources, which include the hostname and port.
URI	Required for most resources	The resource URI. The format of a resource URI is one of the following: - `<BASE URL>/dataflows`. Use this URI to access dataflows,source services, target services,connections, aggregator, links, and transformations. - `<BASE URL>/nodes`. Use this URI to access nodes. - `<BASE_URI>/nodegroups`. Use this URI to access node groups. - `<BASE_URI>/parameters`. Use this URI to access parameters. The Base URL is `http://<hostname>:<port>/administrator/api`

The following table includes the request headers:

Header	Required/ Optional	Description
Content-Type	Required for POST requests	Format of the request. Use the following option: - application/json. Reads request as JSON.
Accept	Optional	Request format that you want. Use the following option: - application/json. Sends response as JSON.
Cookie	Required	Sets the valid session ID as a cookie while a client makes a REST call to any of the resources of EDS. Make the /dtlogin REST call to obtain the session ID. The session ID is set as the JSESSIONID cookie.

Request Body

Use the request body to pass additional attributes for the resource. When you pass attributes in a request body, you pass the attributes as part of an object. You can use the JSON format to pass the attributes.

Return Lists

When the REST API returns a series of objects in JSON, it encloses the list in square brackets.

```
{
    "total": <total number of items>,
    "count": <items fetched in this call>,
    "items": [
        {
            "id": "<ID of the object>",
            "name": "<name of the object>",
            "self": {
                "rel": "self",
                "href": "URI of the object"
            },
            "actions": [
                {
                    "rel": "<action>",
                    "href": "<URI for the action>",
                    "type": "http method"
                }
            ],

            "description": "object DESCRIPTION"
        }
    ]
}
```

REST API Responses

The following table describes the responses to REST API requests:

REST API Request	Successful Response	Failure Response
GET	Returns the requested object, along with the response code 200.	HTTP 404 or 500 error.
POST	Returns the requested object that you created, with a response code 201.	HTTP 400, 404, or 500 error.

Error Object

When the REST API encounters an error, it returns HTTP 400, 404, or 500 error.

For example, an error object might have the following structure:

```
{
 "errors":
[

    {
      "errorType": "NON_RECOVERABLE",
      "code": "400 Bad Request",
      "messageId": "from-below-list",
      "parameters": ["zero-or-more-depending-on-messageId"],
      "errorMessage": "from-below-list"
    }
  ]
}
```

REST API Guidelines

Use the following guidelines when working with EDS REST APIs:

- Use the JSON format to construct a request.
- Specify the format of the request and response in the header. Use the Content-Type attribute to specify the request format and the Accept attribute to specify the response format.
- Use a placeholder for the JSESSIONID in request headers for all resources. Replace the placeholder with the JSESSIONID data returned when you log in to a session.
- For all resources, use a placeholder for the base URL. Replace the placeholder with the Server URL data returned by the login resource.
- All resources and attributes are case sensitive.
- Use the type attribute to define an object in JSON.

CHAPTER 5

HTTP Request and Response Parameters

This chapter includes the following topic:

HTTP Request and Response Parameters Overview

The EDS REST API supports requests and responses in JSON.

Common Request and Response Parameters

The following table describes the common HTTP request and response parameters of the REST calls:

Parameters	Description
id	Unique identifier of the object.
name	The name of the object
valid	The current status of the object.
self	URI reference of the object.
description	Description of the object
type	The object type.
lastModifiedDate	The timestamp when the object was last modified.
createDate	The timestamp when the object was created.
createdBy	The user who created the object.
deployable	Specifies whether the object can be deployed.
deploymentState	The current state of the object, for example, DEPLOYED, NEEDS_DEPLOYMENT, or DRAFT state.

Parameters	Description
force	Force saves an invalid rule or template. If you set the value to true, if the rule or template compilation fails, that rule or template is force saved as an invalid rule or template.
connection	A link to the connection that is referred by the source or target.

Data Flow Parameters

The following table describes the parameters you can use for data flows in the request:

Parameters	Description
name	Name of the data flow.
forceClean	Indicates whether to force clean all data flow configuration from ZooKeeper, for example true or false.

Source Service Parameters

The following table describes the common HTTP request parameters for a source services:

Parameters	Description
name	Name of the source service.
pluginId	Identifier of the source service plug-in. For example, _VDS_SRC_FLAT_FILE, _VDS_SRC_HTTP, _VDS_SRC_JMS, _VDS_SRC_MQTT, _VDS_SRC_STATIC_FILE, _VDS_SRC_SYSLOG_TCP, _VDS_SRC_SYSLOG_UDP, _VDS_SRC_SYSLOG_UDS, _VDS_SRC_TCP, _VDS_SRC_UDP, _VDS_SRC_UDS, _VDS_SRC_SYSLOG_TCP, _VDS_SRC_SYSLOG_UDP, _VDS_SRC_UM, _VDS_SRC_OPC_DA or _VDS_SRC_WS.
config	Configuration for the source service.
stats	Statistics for the source service.

File Source Service

The following table describes the HTTP request parameters for a File source service:

Parameters	Description
directory	Full path to the directory that contains the source files.
fileNameRegex	Exact file name or Java regular expression that matches the active source file name.
isRegexFileName	Indicates that the file name is a regular expression
moveCompletedFilesTo	Location to where the files should be moved after they are processed.
delimiter	End-of-line character sequence that marks the end of a line in the source file. For example, LF, CRLF, or CUSTOM. Default is LF.

Parameters	Description
customDelimiter	Custom character sequence that you want to specify as delimiter for the file source services.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default is 8192. Minimum is 1.

Static File Source Service

The following table describes the HTTP request parameters for a Static File source service:

Parameters	Description
directory	Full path to the directory that contains the source files.
fileNameRegex	Exact file name or Java regular expression that matches the active source file name.
isRegexFileName	Indicates that the file name is a regular expression
noOfThreads	The number of threads used to process the files in parallel. Default is 5. Minimum is 1.
coolingTime	The time in seconds from file modification after which the file should be processed. Specify a waiting time that is greater than the time taken by the application to write to the active file. Default is 5.
moveCompletedFilesTo	Location to where the files should be moved after they are processed.
delimiter	End-of-line character sequence that marks the end of a line in the source file. For example, LF, CRLF, or CUSTOM. Default is LF.
customDelimiter	Custom character sequence that you want to specify as delimiter for the file source service.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

HTTP Source Service

The following table describes the HTTP request parameters for a HTTP or HTTPS source service:

Parameters	Description
connectionType	Connection type. For example, HTTP or HTTPS.
httpPostPath	Path of the HTTP POST requests that you want the HTTP service to receive.
httpsPostPath	Path of the HTTPS POST requests that you want the HTTP service to receive.
keystorePath	Directory that contains the keystore file. Specify the absolute path to file if you select the HTTPS connection type.
keystorePassword	Password for the keystore file. Specify the password if you select the HTTPS connection type.
isPartialPath	Indicate that the path is a prefix.

Parameters	Description
healthCheckUrl	Path to which an HTTP GET request is sent to verify if the server is running.
port	Port on which to listen for incoming connections.
idleTimeout	Time after which the connection is closed when there is no incoming data. Specify a value of 0 to disable timeout.
syncOption	Type of synchronous response. For example, NO_SYNC_RESPONSE, TARGET_ACK_BASED_SYNC_RESPONSE, or HTTP_RESPONSE_BASED_SYNC_RESPONSE. Default is NO_SYNC_RESPONSE.
responseExpiryTime	Time in milliseconds for which the source waits for a POST at the response URL.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

JMS Source Service

The following table describes the HTTP request parameters for a Syslog TCP source service:

Parameters	Description
ctxFactory	The JMS provider specific initial JNDI context factory implementation for connecting to the JNDI service. This value is a fully qualified class name of the Initial Context Factory. Add the JMS Client jar files to the VDS server.
connFactory	The name of the object in the JNDI server that enables the JMS Client to create JMS connections.
providerURL	The location and port of the JMS provider on which to connect.
username	User name to the connection factory.
password	The password of the user account that you use to connect to the connection factory.
destType	The type of destination on which the source service receives JMS messages. For example, TOPIC or QUEUE. Default is TOPIC.
dSubName	Durable subscriptions can receive messages sent while the subscribers are not active. Durable subscriptions provide the flexibility and reliability of queues, but still allow clients to send messages to many recipients. Specify the subscription name if the destination type is TOPIC.
destname	Name of the queue or topic on the JMS Provider as defined in the JMS Connection Factory created by the JMS Administrator. The source service receives JMS messages from this queue or topic.
msgType	Type of source. For example, BYTE, TEXT, or MAP.
ackMode	Acknowledgment mode for non-transacted sessions. You can select one of the following acknowledgment modes: - Auto. The session automatically acknowledges a message when a client receives it. - Client. A client acknowledges a message by calling the message's acknowledge method. Specify AUTO_ACK or CLIENT_ACK. Default is AUTO_ACK

Parameters	Description
clientId	Client identifier to identify the connection and set up a durable connections. To ensure that a publish-subscribe application receives all published messages, use PERSISTENT delivery mode for the publishers. In addition, use durable subscriptions for the subscribers.
selector	Criteria for filtering message header or message properties, to limit which JMS messages the source service receives.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

MQTT Source Service Type

The following table describes the HTTP request parameters for an MQTT source service:

Parameters	Description
url	URL of the MQTT broker or server to which to connect to receive messages. The URL is of the form tcp://<IP_address>:<port>
mqttTopics	Names of the topics to which to subscribe. The MQTT source service supports the topic wildcards that the MQTT specification describes. Specify a comma-separated list of topic names.
clientId	Unique identifier that identifies the connection between the MQTT source service and MQTT broker, and the file-based persistence store that the MQTT source service uses to store messages when they are being processed.
maxEnqueuedMsg	Maximum number of messages that can be stored in the persistence store.
maxMsgPerBatch	Maximum number of messages that are sent by the persistence store in a batch, to a target service.
retryCount	Number of times that the MQTT source service tries to add messages to the internal queue if the internal queue is full.
retryInterval	The time in milliseconds between retries.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

OPC DA Source Service

The following table describes the HTTP request parameters for a OPC DA source service:

Parameters	Description
hostname	Host name of the machine where OPC server is located.
domain	Domain name of the machine where OPC server is located.
username	User name to connect to the machine where the OPC server is located.

Parameters	Description
password	Password to connect to the machine where the OPC server is located.
server-spec-method	URL of the OPC server. The URL can be the programmatic identifier or the class identifier of the OPC DA server.
tag-list-spec-method	Format in which the list of tags is specified. The format can be text or file.
poll-interval	Interval at which the source service reads tags from the OPC DA server.

Syslog TCP Source Service

The following table describes the HTTP request parameters for a Syslog TCP source service:

Parameters	Description
portTCP	TCP port that the source application uses to connect to the Syslog TCP source service.
delimiter	Line feed character sequence that marks the end of a message in the TCP data stream.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

Syslog UDP Source Service

The following table describes the HTTP request parameters for a Syslog UDP source service:

Parameters	Description
portUDP	UDP port that the source application uses to connect to the Syslog UDP source service.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

Syslog UDS Source Service

The following table describes the HTTP request parameters for a Syslog UDS source service:

Parameters	Description
sockAddress	The socket address that the source writes to. The Syslog UDS source service should have write access to the folder that contains the socket file. The socket file is used for communication between the Syslog client and server. For example, /dev/log
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

TCP Source Service

The following table describes the HTTP request parameters for a TCP source service:

Parameters	Description
portTCP	TCP port that the source application uses to connect to the Syslog TCP source service.
delimiter	Line feed character sequence that marks the end of a message in the TCP data stream.
msgLen	Length of each record that is read, in bytes. Specify the length when the source data stream consists of records of a fixed length. The maximum value is 51200.
customDelimiter	Custom character sequence that you want to specify as delimiter for the file source services.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

UDP Source Service

The following table describes the HTTP request parameters for a UDP source service:

Parameters	Description
portUDP	UDP port that the source application uses to connect to the Syslog UDP source service.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

Ultra Messaging Source Service

The following table describes the HTTP request parameters for an Ultra Messaging source service:

Parameter	Description
UMRcv_TopicName	Topic on which the UM sending application publishes messages.
UMRcv_XMLConfig	UM configurations in XML format that the service uses.
UMRcv_AppName	UM sending application name that the source service uses to get the configuration.
ctxName	Context name that the source service uses to get the configuration. If you do not specify a name, the source service gets the configuration from a context with no name.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

WebSocket Source Service

The following table describes the HTTP request parameters for a WebSocket source service:

Parameters	Description
connectionType	Connection type. For example, WS or WSS.
websocketPath	Path of the WebSocket requests that you want the source service to receive.
websocketSecurePath	Path of the WebSocket secure requests that you want the source service to receive.
keystorePath	Directory that contains the keystore file. Specify the absolute path to file if you select the HTTPS connection type.
keystorePassword	Password for the keystore file. Specify the password if you select the WSS connection type.
isPartialPath	Indicate that the path is a prefix.
healthCheckUrl	Path to which an HTTP GET request is sent to verify if the server is running.
port	Port on which to listen for incoming connections.
idleTimeout	Time after which the connection is closed when there is no incoming data. Specify a value of 0 to disable timeout.
eventSize	Maximum length of data that the source service can read at a time, in bytes. Default value is 8192. Minimum value is 1.

Target Service Parameters

The following table describe the common HTTP request parameters for target services:

Parameters	Description
name	Name of the target service.
pluginId	Identifier of the target service plug-in. For example, _VDS_TGT_CASSANDRA , _VDS_TGT_FILE, _VDS_TGT_HDFS, _VDS_TGT_HTTP, _VDS_TGT_JMS, _VDS_TGT_KAFKA, _VDS_TGT_KINESIS, _VDS_TGT_WS, _VDS_TGT_UM, _VDS_TGT_CEP, _VDS_TGT_EVENTHUB or _VDS_TGT_PC
config	Configuration for the target service.
stats	Statistics for the target service.

Cassandra Target Service

The following table describes the HTTP request parameters for a Cassandra target service:

Parameters	Description
contactpoints	URI of the Cassandra database. Use the following URI format: <hostname(s)>:<port>
keyspace	Keyspace to use when writing to the database.
tablename	Table name to use when writing to the database.
username	User name of the Cassandra database.
password	The password of the user account that you use to connect to the Cassandra keyspace.

EventHub Target Service

The following table describes the HTTP request parameters for an EventHub target service:

Parameters	Description
tenantId	The ID of the tenant that the data belongs to. This ID is the Directory ID of the Azure Active Directory.
subscriptionId	The ID of the Azure subscription.
resourceName	The name of the Azure resource group associated with the event hub namespace.
clientApplicationId	The ID of the application created under the Azure Active Directory.
clientSecretKey	The secret key generated for the application.
eventHubNamespace	The name of the Event Hub Namespace that is associated with the resource group name.
eventHubName	The name of the Event Hub.
sharedAccessPolicyName	The name of the Event Hub Namespace Shared Access Policy. To write to an Event hub, the policy must have Send permission.
sharedAccessPolicyKey	The primary key of the Event Hub Namespace Shared Access Policy.

File Target Service

The following table describes the HTTP request parameters for a File target service:

Parameters	Description
path	Full path to the directory that contains the target files.
fileName	Name of the target file. The following special characters are not allowed: * : ? " < > | / \ Maximum length is 200 characters.

Parameters	Description
appendTimestamp	Indicate whether the timestamp must be appended to the file name.
dateFormat	Date and time format to append to the rollover file name. The following special characters are not allowed: * : ? " < > | / \ For example, the following date string represents 3:00 pm on December 07, 2014 if you specify the format as <yyyy>-<MM>-<dd>-<HH>-<mm>-<ss>: 2014-12-07-15-00-00
rolloverSize	Target file size, in megabytes (MB), at which to trigger rollover. A value of zero (0) means that the target service does not roll the file over based on size. Default is 10.
rolloverTime	Length of time, in hours, to keep a target file active. After the time period has elapsed, the target service rolls the file over. A value of zero (0) means that the target service does not roll the file over based on time. Default is 0.
bufferSize	The size of the output stream buffer.
receiveIdleEvents	Indicate whether the target service should flush the data to the file system if the size of the data is less than the buffer size.
idleEventTimeOut	Time in seconds after which the target service flushes the data to the file system.

HDFS Target Service

The following table describes the HTTP request parameters for an HDFS target service:

Parameters	Description
destination	URI of the target file to which the target service writes data. The HDFS target service type supports the following URI formats: HDFS URI format. For example, hdfs://<namenode-name>[:<port>]/<path>/<file-name> MapR URI format. For example, maprfs:///<path>/<file-name>
securityModeList	Indicates if HDFS has Kerberos authentication enabled. For example, NON_SECURE, SECURE. Default is NON_SECURE.
userPrincipal	User principal to log in to the HDFS super user account. Specify a user principal in the following format: user@DOMAIN.COM
keytabPath	Location of the keytab files that HDFS uses. Specify this when the security mode is SECURE.
dateFormat	The time stamp that is appended to the name of the file written to the target. Specify this when the security mode is SECURE.
rolloverSize	Target file size, in gigabytes (GB), at which to trigger rollover. Default is 1.
rolloverTime	Length of time, in hours, to keep a target file active. After the time period has elapsed, the target service rolls the file over. A value of zero (0) means that the HDFS target service does not roll the file over based on time. Default is 0.

Parameters	Description
sync	Flush the client's buffer to the disk device once every second. If you enable forceful synchronization, the data written by the target service is visible to other readers immediately. Specify one of the following values: - hsyncTimeIntervalValue. Time in seconds after which the forceful synchronization occurs. - hsyncBufferLengthValue. The buffer size in bytes after which forceful synchronization occurs.
receiveIdleEvents	Indicate whether the target service should flush the data to the file system if the size of the data is less than the buffer size.
idleEventTimeOut	Time in seconds after which the target service flushes the data to the file system.

HTTP Target Service

The following table describes the HTTP request parameters for an HTTP target service:

Parameters	Description
connectionType	Connection type. For example, HTTP, HTTPS_ALLOW_ALL_CERTIFICATES, or HTTPS_ALLOW_CERTS_IN_TRUSTSTORE. Default is HTTP
targetUrl	URL of the HTTP server. Specify this URL when the connection type is HTTP. Enter the path in the following format: http://<server>:<port>/myapp/path
targetUrlHttpsTrustall	URL of the HTTP server. Specify this URL when the connection type is HTTPS_ALLOW_ALL_CERTIFICATES. Enter the path in the following format: http://<server>:<port>/myapp/path
targetUrlHttpsTruststore	URL of the HTTP server. Specify this URL when the connection type is HTTPS_ALLOW_CERTS_IN_TRUSTSTORE. Enter the path in the following format: http://<server>:<port>/myapp/path
trustStorePath	Path and file name of the Java truststore file. Specify the path if connection type is HTTPS_ALLOW_CERTS_IN_TRUSTSTORE.
trustStorePassword	Password for the truststore file. Specify the password if connection type is HTTPS_ALLOW_CERTS_IN_TRUSTSTORE.

JMS Target Service

The following table describes the HTTP request parameters for a JMS target service:

Parameters	Description
ctxFactory	The JMS provider specific initial JNDI context factory implementation for connecting to the JNDI service. This value is a fully qualified class name of the Initial Context Factory. Add the JMS Client jar files to the VDS server.
connFactory	The name of the object in the JNDI server that enables the JMS Client to create JMS connections.
providerURL	The location and port of the JMS provider on which to connect.

Parameters	Description
username	User name to the connection factory.
password	The password of the user account that you use to connect to the connection factory.

Kafka Target Service

The following table describes the HTTP request parameters for a JMS target service:

Parameters	Description
kafkaDestination	The IP address and port combination of the Kafka messaging system broker.
topic	Topic on which the Kafka target service sends messages.
clientId	An id string to pass to the server when making requests. The purpose of this is to be able to track the source of requests beyond just ip/port by allowing a logical application name to be included in server-side request logging.
securityMode	Indicates if Kafka has Kerberos authentication enabled.
textAreaProducerProps	The properties that are used to connect to Kafka.

Kinesis Target Service

The following table describes the HTTP request parameters for a Kinesis target service:

Parameters	Description
accessKey	Access key ID that the AWS IAM service generates when you create a user.
secretKey	Secret access key that the AWS IAM service generates when you create a user.
streamName	Name of the Kinesis stream to which to write data.
region	Region where the endpoint for your service is available.
parallelRequests	Number of threads in the thread pool. Default is 5.
queueLength	Length of the queue that the target service uses. Default is 10000.
partitionKeyName	Name of the partition key in the message header. For example, IP_ADDRESS, CUSTOM_PARTITION_KEY, or DYNAMIC_PARTITION_KEY. Default is DYNAMIC_PARTITION_KEY.
userDefinedPartitionKey	Name of the custom message header to use as the partition key. Specify this if you use a custom IP address message header as the partition key.

WebSocket Target Service

The following table describes the HTTP request parameters for a WebSocket target service:

Parameters	Description
connectionType	Connection type. For example, WS, WSS_ALLOW_ALL_CERTIFICATES, or WSS_ALLOW_CERTS_IN_TRUSTSTORE. Default is WS.
targetUrl	URL of the WebSocket server. Specify the URL if the connection type is WS. Enter the path in the following format: ws://<server>:<port>/myapp/path
targetUrlWssTrustall	URL of the WebSocket server. Specify the URL if the connection type is WSS_ALLOW_ALL_CERTIFICATES. Enter the path in the following format: wss://<server>:<port>/myapp/path
targetUrlWssTruststore	URL of the WebSocket server. Specify the URL if the connection type is WSS_ALLOW_CERTS_IN_TRUSTSTORE. Enter the path in the following format: wss://<server>:<port>/myapp/path
trustStorePath	Path and file name of the Java truststore file. Specify the path and name if the connection type is WSS_ALLOW_CERTS_IN_TRUSTSTORE.
trustStorePassword	Password for the truststore file. Specify the password if the connection type is WSS_ALLOW_CERTS_IN_TRUSTSTORE.

Transformation Parameters

The following table describes the common HTTP request parameters for transformations and marshallers:

Parameters	Description
name	Name of the transformation.
pluginId	Identifier of the transformation. For example, _VDS_TRX_COMPRESS, _VDS_TRX_DECOMPRESS, _VDS_TRX_INSERT_STR, _VDS_TRX_JS, _VDS_TRX_REGEX, or _VDS_TRX_UDPARSER.
runMode	Indicates whether the transformation runs on the previous entity or the next entity in the data flow. You can specify one the following modes: - WITH_PREVIOUS - WITH_NEXT
config	Configuration for the transformation.
stats	Statistics for the transformation.

Compress Data Transformation

The following table describes the HTTP request parameters for a Compress Data transformation:

Parameters	Description
compressionTechniqueIdx	Compression technique that the transformation type uses to compress data. For example, 0, 1, or 2. Default is 0.

Insert String Transformation

The following table describes the HTTP request parameters for a Insert String transformation:

Parameters	Description
stringExpression	Token expression that describes how to transform each record.

JavaScript Transformation

The following table describes the HTTP request parameters for a JavaScript transformation:

Parameters	Description
js	JavaScript transformation program to transform records and insert delimiters. The program must include the filter function that applies the transformations. Maximum length is 4000 characters.

Regex Filter Transformation

The following table describes the HTTP request parameters for a Regex Filter transformation:

Parameters	Description
regex	Java regular expression to apply to the line of text. For example, you can use the following regular expression: ^.*\[.*\].*$

Unstructured Data Parser Transformation

The following table describes the HTTP request parameters for an Unstructured Data Parser transformation:

Parameters	Description
inputPattern	Pattern that describes how to parse the data. The format must match the pattern defined in the Grok pattern file or the regular expression you specify in Custom Regex.
userRegexes	Regular expression that applies to any part of the input that is not defined in the input pattern.

Aggregator Parameters

The following table describes the parameters in the HTTP request and response content for aggregators:

Parameters	Description
name	Name of the aggregator.
pluginId	Identifier of the aggregator plug-in. Specify _VDS_AGGREGATOR
config	Configuration for the aggregator.
eventQueueSize	Maximum number of events that the aggregator stores in an in-memory queue until the events are consumed by the target.
stats	Statistics for the aggregator.

Connection Parameters

The following table describes the parameters in the HTTP request and response content for connections:

Parameters	Description
name	Name of the connection.
type	Type of entity. For example, _VDS_UM_TRANSPORT or _VDS_WS_TRANSPORT.

Ultra Messaging Transport

The following table describes the parameters in the HTTP request content for the Ultra Messaging transport:

Parameters	Description
resolution	Type of connection. Specify unicast or multicast topic resolution type.
umMode	Mode in which the source service distributes data to the target service. For example, ULB, UMS, or, UMP.
umConnectionType	Indicates if the connection is secure or nonsecure.
certificate	Certificate file name. Provide the file name if the connection is secure.
certificateKey	Key file name. Provide the file name if the connection is secure.
certificateKeyPassword	Password to the key file. Provide the password if the connection is secure.
trustedCertificates	Truststore file name. Provide the file name if the connection is secure.
daemon	Address of the topic resolution daemon. Provide the daemon address if the topic resolution is unicast.
address	Multicast address that you want to use. Provide the address if the topic resolution is multicast.

Parameters	Description
resolverPort	The unicast or multicast UDP port.
requestTcpPort	The port on which to listen for responses from requests.
transportTcpPort	The preferred TCP port for the topic that the UM application publishes on.
umConfig	The UM configuration that the data connection uses.

WebSocket Transport

The following table describes the parameters in the HTTP request and response content for the WebSocket transport:

Parameters	Description
messagingMode	Mode in which the source service distributes data to the target service. For example, ACKNOWLEDGEMENT or STREAMING
wsConnectionType	Indicates if the connection is secure or nonsecure. For example, WS (for WebSocket), or WSS (for WebSocket Secure). Default is WS.
keystorePath	Keystore file name. Provide the file name if the connection is secure.
keystorePassword	Password to the keystore file name. Provide the file name if the connection is secure.
truststorePath	Truststore file name. Provide the password if the connection is secure.
truststorePassword	Password to the truststore file name. Provide the file name if the connection is secure.
port	Port on which the target service listens for incoming connections.
healthCheckUrl	Path to which an HTTP GET request is sent to verify if the server is running.

Link Parameters

The following table describes the parameters in the HTTP request and response content for links:

Parameters	Description
fromId	Identifier of the entity which you are linking from.
toIds	Identifier or entities to which you are linking to. If there is only one entity, the link is a point-to-point link. If there are many entities, the link is a publish-subscribe link.
connectionID	Identifier of the connection.
transportTopic	Topic name of the transport on which the source service publishes data. This topic is known as well-known topic.

Parameters	Description
topicIds	List of topic identifiers.
reponseIds	List of response identifiers.

Node Group Parameters

The following table describes the HTTP request parameters for a node group:

Attributes	Description
name	Name of the node group.
type	Type of node group. For example, DYNAMIC or STATIC.
pattern	Regular expression pattern for the dynamic node group.
nodes	List of node identifiers associated with the node group.

Node Parameters

The following table describes the parameters in the HTTP request content for node groups:

Parameters	Description
name	Name of the node.
nodeGroups	List of node group identifiers associated with the node if the node group is static.

EDS Parameter Parameters

The following table describes the parameters you can use for EDS parameters in the request:

Parameters	Description
key	Name of the parameter.
value	Value for the parameter.
secure	Indicates if the parameter is secure.

CHAPTER 6

Sample JSON Requests and Responses

This chapter includes the following topics:

Data Flows

You can use API calls to perform various operations with data flows.

Create Data Flows

Use the POST method to create a data flow in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a data flow:

```
POST /api/dataflows

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
"name": "Test Dataflow",
"description":"new dataflow"
}
```

POST Response in JSON Format

The following is a sample response for the 201 Created code:

```
{
    "id": {dataflow-id},
    "name": "Test Dataflow",
                "description":"new dataflow"
    "success": true,
    "deploymentState": "DRAFT"
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
            [
                {
                "code": INVALID_DATAFLOW_NAME,
                "parameters": null,
                "errorMessage": "Dataflow name not valid."
                }
            ]
}
```

```
{
  "success": false,
  "errors":
        [
            {
            "code": DATAFLOW_NAME_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Dataflow name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to create the dataflow. Exception : {error-message}"
            }
        ]
}
```

Retrieve Data Flows

Use the GET method to retrieve data flows in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a data flow:

```
GET /api/dataflows/{dataflow-id}
```

```
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "success": true,
    "id": {dataflow-id},
    "name": "Dataflow 1",
    "deploymentState": "DRAFT",
    "entities": {
        "count": 3,
        "total": 3,
        "items": [
            {
                "id": 1,
                "name": "TCP",
                "type": "SOURCE",
                "pluginId": "_VDS_SRC_TCP",
                "deploymentState": "DRAFT"
            }, {
                "id": 2,
                "name": "Ultra Messaging",
                "type": "TARGET",
                "pluginId": "_VDS_TGT_UM",
                "deploymentState": "DRAFT"
            }, {
                "id": 3,
                "name": "um",
                "type": "connection",
                "pluginId": "_VDS_UM_TRANSPORT",
                "deploymentState": "DRAFT"
            }
        ]
    }
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get dataflow. Exception: {error-message}"
            }
        ]
}
```

Retrieve All Data Flows

Use the GET method to retrieve all data flows in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all data flows:

```
GET /api/dataflows
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is the sample response for the 200 OK code when there are no data flows:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}
```

The following is the sample response for the 200 OK code when there are three data flows:

```
{
    "success": true,
    "total": 3,
    "count": 3,
    "items": [
                { "id": 1, "name": "Dataflow 1", "deploymentState": "DRAFT" },
                { "id": 2, "name": "My second Dataflow", "deploymentState": "DEPLOYED" },
                { "id": 3, "name": "DF3", "deploymentState": "NEEDS_REDEPLOYMENT" }
            ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all dataflows. Exception: {error-message}"
            }
        ]
}
```

Update Data flows

Use the POST method to update a data flow in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a data flow:

```
POST /api/dataflows/{dataflow-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following a sample POST body in JSON format:

```
{
    "name": "Test Dataflow Updated",
```

```
                "description":"name updated"
}
```

PUT Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id" : {dataflow-id}
    "name": {updated-dataflow-name}
                "description":"name updated",
    "success": true,
    "deploymentState": "DRAFT"
}
```

The following is the sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INVALID_DATAFLOW_NAME,
            "parameters": null,
            "errorMessage": "Dataflow name not valid."
            }
        ]
}


{
    success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NAME_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Dataflow name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update dataflow. Exception : {error-message}."
            }
        ]
}
```

Deploy Data Flows

Use the POST method to deploy a data flow in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deploys a data flow:

```
POST /api/dataflows/{dataflow-id}/deploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "success": true,
    "name" : {dataflow-name},
    "id": {dataflow-id},
    "deploymentState": "DEPLOYED"
}
```

The following is the sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is a sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NO_CONNECTION_BETWEEN_ENTITIES,
            "parameters": null,
            "errorMessage": "The deploy action failed because there is no connection
between entities. Connect source and target entities using connection.",
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            errorMessage": "Failed to deploy dataflow. Exception : {error-message}."
            }
        ]
}
```

Deploy All Data Flows

Use the POST method to deploy all data flows in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deploys a data flow:

```
POST /api/dataflows/deployAll

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "success": true
}
```

The following is a sample response for the 500 Internal Server Error code

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to deploy all dataflows. Exception: {error-
message}.",
            }
        ]
}
```

Undeploy Data Flows

Use the POST method to undeploy a data flow in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request undeploys a data flow:

```
POST /api/dataflows/{dataflow-id}/undeploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "name" : {dataflow-name},
    "success": true,
    "id": {dataflow-id},
    "deploymentState": "DRAFT"
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
```

```
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    success": false,
    errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to undeploy dataflow. Exception : {error-message}."
            }
        ]
}
```

Undeploy All Data Flows

Use the POST method to undeploy all data flows in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deploys a data flow:

```
POST /api/dataflows/undeployAll

Content-Type: application/json
Accept: application/json
```

POST body in JSON format

The body can be one of the following:

```
{
    "forceClean": true
}
```

or

```
{
    "forceClean": false
}
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "success": true
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,,
            "parameters": [{error-message}],
            "errorMessage": "Failed to undeploy all dataflows. Exception: {error-
message}."
            }
        ]
}
```

Delete Data Flows

Use the POST method to delete a data flow in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a data flow:

```
POST /api/dataflows/{dataflow-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors": [
        {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
        }
    ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update dataflow. Exception : {error-message}."
            }
        ]
}
```

Redeploy Data Flows

Use the POST method to redeploy a data flow in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request redeploys a data flow:

```
POST /api/dataflows/redeploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "name" : {dataflow-name},
    "success": true,
    "id": {dataflow-id},
    "deploymentState": "DEPLOYED"
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to redeploy dataflow. Exception: {error-message}.",
            }
        ]
}
```

Source Services

You can use API calls to perform various operations with source services.

Create Sources

Use the POST method to create a source in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a source:

```
POST /api/dataflows/{dataflow-id}/sources

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
    "name":"TestSysLogTCP",
    "pluginId":"_VDS_SRC_SYSLOG_TCP",
    "config":
    {
        "portTCP":"7896","delimiter":"LF","eventSize":"8192","persistEvents":"false",
        "persistOptions":"# --\n# The interval in milliseconds after which the writes
\n# are flushed to disk.\n# --\n#flushInterval=5000\n\n# --\n# The maximum number of db
files to keep\n# --\n#maxDataFiles=10\n\n# --\n# Size of individual db file in bytes
after which the files are rolled over\n# --\n#maxDataFileSize=1073741824\n\n# --\n# The
max size of unsent data in bytes.    If 0, it is unbounded\n# --
\n#maxQueueSizeInBytes=0\n\n# --\n# The batch size in bytes up to which the data is
buffered before writing to the disk.\n# --\n#batchSizeInBytes=262144"
    },
    "stats":
    [
            {"name": "Bytes Sent","value": 10001,"type": 101}, {"name": "Events
Sent","value": 10003,"type": 101}, {"name": "Events to be Sent","value": 10006,"type":
```

```
101},
            {"name": "Events not Delivered","value": 10002,"type": 101}, {"name": "Send
Rate(Per Sec)","value": 10005,"type": 101}, {"name":"Events Dropped","value":1,"type":
103},
            {"name":"Concurrent Connections","value":2,"type":103}, {"name":"Maximum
Concurrent Clients","value":3,"type":103}
    ]
}
```

POST Response in JSON Format

The following is a sample response for the 201 Created code:

```
{
    "id": {source-entity-id},
    "name": "TestSysLogTCP",
    "type": "SOURCE",
    "self": {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/sources/{source-entity-id}",
        "title": "TestSysLogTCP",
        "id": {source-entity-id}
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
        "title": "TestDataflow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-06T12:49:47+05:30",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "pluginId": "_VDS_SRC_SYSLOG_TCP",
    "schedulable": false,
    "config":
    {
        "portTCP": "7896",
        "delimiter":"0",
        "eventSize": "8192",
        "persistEvents": "false",
        "persistOptions":"# --\n# The interval in milliseconds after which the writes
\n# are flushed to disk.\n# --\n#flushInterval=5000\n\n# --\n# The maximum number of db
files to keep\n# --\n#maxDataFiles=10\n\n# --\n# Size of individual db file in bytes
after which the files are rolled over\n# --\n#maxDataFileSize=1073741824\n\n# --\n# The
max size of unsent data in bytes. If 0, it is unbounded\n# --\n#maxQueueSizeInBytes=0\n
\n# --\n# The batch size in bytes up to which the data is buffered before writing to the
disk.\n# --\n#batchSizeInBytes=262144"
    },
    "stats":
    [
        {"name": "Bytes Sent","value": 10001,"type": 101}, {"name": "Events
Sent","value": 10003,"type": 101}, {"name": "Events to be Sent","value": 10006,"type":
101},
        {"name": "Events not Delivered","value": 10002,"type": 101}, {"name": "Send
Rate(Per Sec)","value": 10005,"type": 101}, {"name":"Events Dropped","value":1,"type":
103},
        {"name":"Concurrent Connections","value":2,"type":103}, {"name":"Maximum
Concurrent Clients","value":3,"type":103}
    ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
```

```
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is a sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to create source. Exception: {error-message}"
            }
        ]
}
```

Get Source by ID

Use the GET method to retrieve sources based on ID in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a source based on the ID:

```
GET /api/dataflows/{dataflow-id}/sources/{source-entity-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {source-entity-id},
    "name": "TestSysLogTCP",
    "type": "SOURCE",
    "pluginId": "_VDS_SRC_SYSLOG_TCP",
    "self": {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/sources/{source-entity-id}",
        "title": "TestSysLogTCP",
        "id": {source-entity-id}
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
```

```
        "title": "TestDataflow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-06T14:55:48+05:30",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "schedulable": false,
    "config":
    {
        "delimiter": "0",
        "eventSize": "8192",
        "persistEvents": "false",
        "persistOptions": "# --\n# The interval in milliseconds after which the writes
\n# are flushed to disk.\n# --\n#flushInterval=5000\n\n# --\n# The maximum number of db
files to keep\n# --\n#maxDataFiles=10\n\n# --\n# Size of individual db file in bytes
after which the files are rolled over\n# --\n#maxDataFileSize=1073741824\n\n# --\n# The
max size of unsent data in bytes. If 0, it is unbounded\n# --\n#maxQueueSizeInBytes=0\n
\n# --\n# The batch size in bytes up to which the data is buffered before writing to the
disk.\n# --\n#batchSizeInBytes=262144"
        },
    "stats":
    [
        {"name": "Bytes Sent","value": 10001,"type": 101}, {"name": "Events
Sent","value": 10003,"type": 101}, {"name": "Events to be Sent","value": 10006,"type":
101},
        {"name": "Events not Delivered","value": 10002,"type": 101}, {"name": "Send
Rate(Per Sec)","value": 10005,"type": 101}, {"name":"Events Dropped","value":1,"type":
103},
        {"name":"Concurrent Connections","value":2,"type":103}, {"name":"Maximum
Concurrent Clients","value":3,"type":103}
    ]
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}]
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": SOURCE_ENTITY_NOT_FOUND,
            "parameters": [{source-entity-id}],
            "errorMessage": "Source entity with Id: {source-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
```

```
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get source. Exception: {error-message}"
            }
        ]
}
```

Retrieve All Sources

Use the GET method to retrieve all sources in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all sources:

```
GET /api/dataflows/{dataflow-id}/sources
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is the sample response for the 200 OK code when there are no sources:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}
```

The following is a sample response when there is one source in the data flow:

```
{
    "total": 1,
    "count": 1,
    "success": true,
    "items": [
        {
            "id": {source-entity-id},
            "name": "TestSysLogTCP",
            "type": "SOURCE",
            "pluginId": "_VDS_SRC_SYSLOG_TCP",
            "self": {
                "rel": "self",
                "href": "api/dataflows/{dataflow-id}/sources/{source-entity-id}",
                "title": "TestSysLogTCP",
                "id": {source-entity-id}
            },
            "dataflow": {
                "rel": "dataflow",
                "href": "api/dataflows/{dataflow-id}",
                "title": "TestDataflow",
                "id": {dataflow-id}
            },
            "valid": true,
            "createDate": "2015-07-06T14:55:48+05:30",
            "createdBy": "Administrator",
            "predefined": false,
            "deployable": true,
            "deploymentState": "DRAFT",
            "force": false,
            "schedulable": false,
            "config":
            {
                "delimiter": "0",
                "eventSize": "8192",
                "persistEvents": "false",
                "persistOptions": "# --\n# The interval in milliseconds after which the
writes \n# are flushed to disk.\n# --\n#flushInterval=5000\n\n# --\n# The maximum number
```

```
of db files to keep\n# --\n#maxDataFiles=10\n\n# --\n# Size of individual db file in
bytes after which the files are rolled over\n# --\n#maxDataFileSize=1073741824\n\n# --
\n# The max size of unsent data in bytes. If 0, it is unbounded\n# --
\n#maxQueueSizeInBytes=0\n\n# --\n# The batch size in bytes up to which the data is
buffered before writing to the disk.\n# --\n#batchSizeInBytes=262144"
            },
            "stats":
            [
                {"name": "Bytes Sent","value": 10001,"type": 101}, {"name": "Events
Sent","value": 10003,"type": 101}, {"name": "Events to be Sent","value": 10006,"type":
101},
                {"name": "Events not Delivered","value": 10002,"type": 101}, {"name":
"Send Rate(Per Sec)","value": 10005,"type": 101}, {"name":"Events Dropped","value":
1,"type":103},
                {"name":"Concurrent Connections","value":2,"type":103}, {"name":"Maximum
Concurrent Clients","value":3,"type":103}
            ]
        }
    ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all sources. Exception: {error-message}"
            }
        ]
}
```

Update Source Services

Use the POST method to update a source service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a source service:

```
POST /api/dataflows/{dataflow-id}/sources/{source-entity-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following a sample POST body in JSON format:

```
{
    "name":"TestSysLogTCPChanged",
    "pluginId":"_VDS_SRC_SYSLOG_TCP",
```

```
    "config":
    {
        "portTCP":"8989","delimiter":"LF","eventSize":"8192","persistEvents":"false",
        "persistOptions":"# --\n# The interval in milliseconds after which the writes
\n# are flushed to disk.\n# --\n#flushInterval=5000\n\n# --\n# The maximum number of db
files to keep\n# --\n#maxDataFiles=10\n\n# --\n# Size of individual db file in bytes
after which the files are rolled over\n# --\n#maxDataFileSize=1073741824\n\n# --\n# The
max size of unsent data in bytes.    If 0, it is unbounded\n# --
\n#maxQueueSizeInBytes=0\n\n# --\n# The batch size in bytes up to which the data is
buffered before writing to the disk.\n# --\n#batchSizeInBytes=262144"
    },
    "stats":
    [
        {"name": "Bytes Sent","value": 10001,"type": 101}, {"name": "Events
Sent","value": 10003,"type": 101}, {"name": "Events to be Sent","value": 10006,"type":
101},
        {"name": "Events not Delivered","value": 10002,"type": 101}, {"name": "Send
Rate(Per Sec)","value": 10005,"type": 101}, {"name":"Events Dropped","value":1,"type":
103},
        {"name":"Concurrent Connections","value":2,"type":103}, {"name":"Maximum
Concurrent Clients","value":3,"type":103}
    ]
}
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {source-entity-id},
    "name": "TestSysLogTCPChanged",
    "type": "SOURCE",
    "self": {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/sources/{source-entity-id}",
        "title": "TestSysLogTCP",
        "id": {source-entity-id}
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
        "title": "TestDataflow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-06T12:49:47+05:30",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "pluginId": "_VDS_SRC_SYSLOG_TCP",
    "schedulable": false,
    "config":
    {
        "portTCP": "8989",
        "delimiter":"0",
        "eventSize": "8192",
        "persistEvents": "false",
        "persistOptions":"# --\n# The interval in milliseconds after which the writes
\n# are flushed to disk.\n# --\n#flushInterval=5000\n\n# --\n# The maximum number of db
files to keep\n# --\n#maxDataFiles=10\n\n# --\n# Size of individual db file in bytes
after which the files are rolled over\n# --\n#maxDataFileSize=1073741824\n\n# --\n# The
max size of unsent data in bytes. If 0, it is unbounded\n# --\n#maxQueueSizeInBytes=0\n
\n# --\n# The batch size in bytes up to which the data is buffered before writing to the
disk.\n# --\n#batchSizeInBytes=262144"
    },
    "stats":
    [
        {"name": "Bytes Sent","value": 10001,"type": 101}, {"name": "Events
```

```
Sent","value": 10003,"type": 101}, {"name": "Events to be Sent","value": 10006,"type":
101},
        {"name": "Events not Delivered","value": 10002,"type": 101}, {"name": "Send
Rate(Per Sec)","value": 10005,"type": 101}, {"name":"Events Dropped","value":1,"type":
103},
        {"name":"Concurrent Connections","value":2,"type":103}, {"name":"Maximum
Concurrent Clients","value":3,"type":103}
    ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update source. Exception: {error-message}"
            }
        ]
}
```

Delete Source Services

Use the POST method to delete a source service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a source service:

```
POST /api/dataflows/{dataflow-id}/sources/{source-entity-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors": [
        {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
        }
    ]
}
```

```
{
    "success": false,
    "errors":
        [
            {
            "code": SOURCE_ENTITY_NOT_FOUND,
            "parameters": [{source-entity-id}],
            "errorMessage": "Source Entity with Id: {source-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update dataflow. Exception : {error-message}."
            }
        ]
}
```

Associate Source Services with Node Groups

Use the POST method to associate a source service with node groups in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request associates source services with node groups:

```
POST /api/dataflows/<dataflow-id>/sources/<source-id>/associatenodegroups

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
  "nodeGroups":[nodegroupId]
}
```

```
{
            "nodeGroups":[107]
}
```

POST Response in JSON Format

The following is a sample response:

```
 "success": true,
  "total": 1,
```

```
  "count": 1,
  "items": [
    {
      "id": 107,
      "name": "nodegroup1",
      "success": true,
      "type": "STATIC",
      "nodes": [
        {
          "id": 106,
          "name": "node1"
        }
      ]
    }
  ]
}
```

Retrieve Source Service and Node Group Associations

Use the GET method to retrieve source service and node group associations in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a source service and node group association:

```
GET /api/dataflows/{dataflow-id}/sources/{source-entity-id}/associatenodegroups

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response:

```
{ "success": true,
  "total": 1,
  "count": 1,
  "items": [
    { "id": 1, "name": "ng1",
      "nodes": [
        { "id": 1, "name": "node10" },
        { "id": 2, "name": "node20" }
      ]
    }
  ]
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": SOURCE_ENTITY_NOT_FOUND,
            "parameters": [{source-entity-id}],
```

```
            "errorMessage": "Source Entity with Id: {source-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get nodegroups associated with source. Exception:
{error-message}"
            }
        ]
}
```

Deploy Source Services

Use the POST method to deploy a source service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deploys a source service:

```
POST /api/dataflows/{dataflow-id}/sources/{source-entity-id}/deploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}


{
    "success": false,
    "errors":
        [
            {
            "code": SOURCE_ENTITY_NOT_FOUND,
            "parameters": [{source-entity-id}],
            "errorMessage": "Source Entity with Id: {source-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
```

```
                "errorMessage": "Failed to deploy entity. Exception: {error-message}"
                }
            ]
}
```

Undeploy Source Services

Use the POST method to undeploy a source service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request undeploys a source service:

```
POST /api/dataflows/{dataflow-id}/sources/{source-entity-id}/undeploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": SOURCE_ENTITY_NOT_FOUND,
            "parameters": [{source-entity-id}],
            "errorMessage": "Source Entity with Id: {source-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to undeploy entity. Exception: {error-message}"
            }
        ]
}
```

Redeploy Source Services

Use the POST method to redeploy a source service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request redeploys a source service:

```
POST /api/dataflows/{dataflow-id}/sources/{source-entity-id}/redeploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": SOURCE_ENTITY_NOT_FOUND,
            "parameters": [{source-entity-id}],
            "errorMessage": "Source Entity with Id: {source-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to redeploy entity. Exception: {error-message}"
            }
        ]
}
```

Target Services

You can use API calls to perform various operations with target services.

Create Target Services

Use the POST method to create a target service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a target service:

```
POST /api/dataflows/{dataflow-id}/targets

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
Content-Type: application/json
Accept: application/json

{
    "name": "TestUMTarget",
    "pluginId": "_VDS_TGT_UM",
    "config": {},
    "stats": []
}
```

POST Response in JSON Format

The following is a sample response for the 201 Created code:

```
{
    "id": {target-entity-id},
    "name": "TestUMTarget",
    "type": "TARGET",
    "pluginId": "_VDS_TGT_UM",
    "self": {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/targets/{target-entity-id}",
        "title": "TestUMTarget",
        "id": {target-entity-id}
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
        "title": "Data flow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-07T03:21:39-07:00",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "schedulable": false,
    "config": {},
    "stats": []
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
```

```
        ]
}
```

The following is a sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to create target. Exception: {error-message}"
            }
        ]
}
```

Get Target Services by ID

Use the GET method to retrieve target services based on ID in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a target service based on the ID:

```
GET /api/dataflows/{dataflow-id}/targets/{target-entity-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {target-entity-id},
    "name": "TestUMTarget",
    "type": "TARGET",
    "self": {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/targets/{target-entity-id}",
        "title": "TestUMTarget",
        "id": {target-entity-id}
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
        "title": "Data flow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-07T03:21:39-07:00",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
```

```
    "deploymentState": "DRAFT",
    "force": false,
    "pluginId": "_VDS_TGT_UM",
    "schedulable": false,
    "config": {},
    "stats": []
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

```
{
    "success": false,
    "errors":
        [
            {
            "code": TARGET_ENTITY_NOT_FOUND,
            "parameters": [{target-entity-id}],
            "errorMessage": "Target entity with Id: {target-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get target. Exception: {error-message}"
            }
        ]
}
```

Retrieve All Target Services

Use the GET method to retrieve all target services in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all target services:

```
GET /api/dataflows/{dataflow-id}/targets

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is the sample response for the 200 OK code when there are no targets:

```
{ "success": true,
  "total": 0,
  "count": 0,
  "items": []
}
```

The following is a sample response when there is one UM target in the data flow:

```
{
    "total": 1,
    "count": 1,
    "success": true,
    "items": [
        {
            "id": {target-entity-id},
            "name": "um-target",
            "type": "TARGET",
            "pluginId": "_VDS_TGT_UM",
            "self": {
                "rel": "self",
                "href": "api/dataflows/{dataflow-id}/targets/{target-entity-id}",
                "title": "um-target",
                "id": {target-entity-id}
            },
            "dataflow": {
                "rel": "dataflow",
                "href": "api/dataflows/{dataflow-id}",
                "title": "Data flow",
                "id": {dataflow-id}
            },
            "valid": true,
            "createDate": "2015-07-07T02:31:41-07:00",
            "createdBy": "Administrator",
            "predefined": false,
            "deployable": true,
            "deploymentState": "DRAFT",
            "force": false,
            "schedulable": false,
            "config": {},
            "stats": []
        }
    ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all targets. Exception: {error-message}"
            }
        ]
}
```

Update Target Services

Use the POST method to update a target service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a target service:

```
POST /api/dataflows/{dataflow-id}/targets/{target-entity-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following a sample POST body in JSON format:

```
{
    "config": {},
    "stats": [],
    "name": "TestUMTargetUpdated",
    "type":"target",
    "pluginId": "_VDS_TGT_UM"
}
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {target-entity-id},
    "name": "TestUMTargetUpdated",
    "type": "TARGET",
    "self": {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/targets/{target-entity-id}",
        "title": "TestUMTargetUpdatedAgain",
        "id": {target-entity-id}
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
        "title": "Data flow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-07T03:21:39-07:00",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "pluginId": "_VDS_TGT_UM",
    "schedulable": false,
    "config": {},
    "stats": []
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
```

```
            }
        ]
}
```

The following is sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update target. Exception: {error-message}"
            }
        ]
}
```

Delete Target Services

Use the POST method to delete a target service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a target service:

```
POST /api/dataflows/{dataflow-id}/targets/{target-entity-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": TARGET_ENTITY_NOT_FOUND,
            "parameters": [{target-entity-id}],
```

```
            "errorMessage": "Target Entity with Id: {target-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to delete target. Exception: {error-message}"
            }
        ]
}
```

Associate Target Services with Node Groups

Use the POST method to update a target service and node group associations or to associate target services with node groups in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request associates a target service with a node group:

```
POST /api/dataflows/{dataflow-id}/targets/{target-entity-id}/associatenodegroups

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
  "nodeGroups": [ 1 ]
}
```

POST Response in JSON Format

The following is a sample response:

```
{ "success": true,
  "total": 1,
  "count": 1,
  "items": [
    { "id": 1, "name": "ng1",
      "nodes": [
        { "id": 1, "name": "node10" },
        { "id": 2, "name": "node20" }
      ]
    }
  ]
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
```

```
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": TARGET_ENTITY_NOT_FOUND,
            "parameters": [{target-entity-id}],
            "errorMessage": "Target Entity with Id: {target-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to associate nodegroup to target. Exception: {error-
message}"
            }
        ]
}
```

Retrieve Target Service and Node Group Associations

Use the GET method to retrieve target service and node group associations in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a target service and node group association:

```
GET /api/dataflows/{dataflow-id}/targets/{target-entity-id}/associatenodegroups

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response:

```
{ "success": true,
  "total": 1,
  "count": 1,
  "items": [
    { "id": 1, "name": "ng1",
      "nodes": [
        { "id": 1, "name": "node10" },
        { "id": 2, "name": "node20" }
      ]
    }
  ]
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
```

```
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": TARGET_ENTITY_NOT_FOUND,
            "parameters": [{target-entity-id}],
            "errorMessage": "Target Entity with Id: {target-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get nodegroups associated with target. Exception:
{error-message}"
            }
        ]
}
```

Deploy Target Services

Use the POST method to deploy a target service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deploys a target service:

```
POST /api/dataflows/{dataflow-id}/targets/{target-entity-id}/deploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": TARGET_ENTITY_NOT_FOUND,
            "parameters": [{target-entity-id}],
            "errorMessage": "Target Entity with Id: {target-entity-id} not found"
            }
```

```
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to deploy entity. Exception: {error-message}"
            }
        ]
}
```

Undeploy Target Services

Use the POST method to undeploy a target service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request undeploys a target service:

```
POST /api/dataflows/{dataflow-id}/targets/{target-entity-id}/undeploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": TARGET_ENTITY_NOT_FOUND,
            "parameters": [{target-entity-id}],
            "errorMessage": "Target Entity with Id: {target-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to undeploy entity. Exception: {error-message}"
            }
        ]
}
```

Redeploy Target Services

Use the POST method to redeploy a target service in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request redeploys a target service:

```
POST /api/dataflows/{dataflow-id}/targets/{target-entity-id}/redeploy

Content-Type: application/json
Accept: application/json
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": TARGET_ENTITY_NOT_FOUND,
            "parameters": [{target-entity-id}],
            "errorMessage": "Target Entity with Id: {target-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to redeploy entity. Exception: {error-message}"
            }
        ]
}
```

Data Connections

You can use API calls to perform various operations with data connections.

Create Data Connections

Use the POST method to create a data connection in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a data connection:

```
POST /api/dataflows/{dataflow-id}/connections
Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

The following is the sample POST body for a nonsecure Ultra Messaging connection:

```
{
    "name": "SimpleUMConnection",
    "resolution": "UNICAST",
    "address": "",
    "daemon": "127.0.0.1:15380",
    "umMode": "ULB",
    "umConnectionType": "UM",
    "umpStoreName": "",
    "resolverPort": "",
    "requestTcpPort": "",
    "transportTcpPort": "",
    "umConfig": "",
    "type": "_VDS_UM_TRANSPORT"
}
```

The following is the sample POST body for a secure Ultra Messaging connection:

```
{
    "name": "SecureUMConnection",
    "resolution": "UNICAST",
    "address": "",
    "daemon": "127.0.0.1:15380",
    "umMode": "ULB",
    "umConnectionType": "UMSECURE",
    "cipherSuites": "",
    "certificate": "/data1/certs/cert.crt",
    "certificateKey": "/data1/certs/cert.key",
    "certificateKeyPassword": "${certKeyPassword}",
    "trustedCertificates": "/data1/certs/cert.crt",
    "umpStoreName": "",
    "resolverPort": "",
    "requestTcpPort": "",
    "transportTcpPort": "",
    "umConfig": "",
    "type": "_VDS_UM_TRANSPORT"
}
```

The following is the sample POST body for a WebSocket connection:

```
{
    "name": "SampleWSConnection",
    "messagingMode": "ACKNOWLEDGEMENT",
    "wsConnectionType": "WS",
    "keystorePath": "",
    "keystorePassword": "",
    "truststorePath": "",
    "truststorePassword": "",
    "port": "1234",
    "healthCheckUrl": "status",
    "server": "",
    "advancedConfig": "",
    "type": "_VDS_WS_TRANSPORT"
}
```

POST Response in JSON Format

The following are sample responses for the 201 Created code:

```
{{
    "id": {connection-id},
    "name": "Sample_UM_Connection",
    "success": true,
    "type": "_VDS_UM_TRANSPORT",
    "resolution": "UNICAST",
    "address": "",
    "daemon": "127.0.0.1:15380",
    "umMode": "ULB",
    "umpStoreName": "",
    "resolverPort": "",
    "requestTcpPort": "",
    "transportTcpPort": "",
    "umConfig": ""
}

{
    "id": {connection-id},,
    "name": "SampleWSConnection",
    "success": true,
    "type": "_VDS_WS_TRANSPORT",
    "wsMode": "ACKNOWLEDGEMENT",
    "secure": false,
    "keystorePath": "",
    "keystore_password": "",
    "truststorePath": "",
    "truststorePassword": "",
    "port": "1234",
    "healthCheckUrl": "status",
    "server": "",
    "advancedConfig": ""
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": CONNECTION_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Connection name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
```

```
                "errorMessage": "Failed to create connection. Exception: {error-message}"
                }
            ]
}
```

Retrieve Data Connections

Use the GET method to retrieve data connections in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a data connection:

```
GET /api/dataflows/{dataflow-id}/connections/{connection-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {connection-id},
    "name": "UM_transport_1",
    "type": "_VDS_UM_TRANSPORT",
    "resolution": "UNICAST",
    "address": "",
    "daemon": "127.0.0.1:15380",
    "umMode": "ULB",
    "umpStoreName": "",
    "resolverPort": "",
    "requestTcpPort": "",
    "transportTcpPort": "",
    "umConfig": ""
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": CONNECTION_NOT_FOUND,
            "parameters": [{connection-id}],
            "errorMessage": "Connection with Id: {connection-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get connection. Exception : {error-message}."
            }
        ]
}
```

Retrieve All Data Connections

Use the GET method to retrieve all data connections in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all data connections:

```
GET /api/dataflows/{dataflow-id}/connections
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following are sample responses for the 200 OK code when there are no data flows:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}
```

```
{
    "total": 2,
    "count": 2,
    "success": true,
    "items": [
        {
            "id": {connection-id-1},
            "name": "UM_transport_1",
            "type": "_VDS_UM_TRANSPORT",
            "resolution": "UNICAST",
            "address": "",
            "daemon": "127.0.0.1:15380",
            "umMode": "UMS",
            "umpStoreName": "",
            "resolverPort": "",
            "requestTcpPort": "",
            "transportTcpPort": "",
            "umConfig": ""
        },
        {
            "id": {connection-id-2},,
            "name": "WS_transport_1",
            "type": "_VDS_WS_TRANSPORT",
            "wsMode": "ACKNOWLEDGEMENT",
            "secure": false,
            "keystorePath": "",
            "keystore_password": "",
            "truststorePath": "",
            "truststorePassword": "",
            "port": "1234",
            "healthCheckUrl": "status",
            "server": "",
            "advancedConfig": ""
        }
    ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all connections. Exception : {error-message}."
            }
```

```
        ]
}
```

Update Data Connections

Use the POST method to update data connections in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a data connection:

```
POST /api/dataflows/{dataflow-id}/connections/{connection-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following is a sample POST body in JSON format for an Ultra Messaging connection:

```
{
    "name":"um_connection1_updated",
    "type":"_VDS_UM_TRANSPORT",
    "resolution":"UNICAST",
    "daemon":"127.0.0.1:15380",
    "umMode":"ULB"
}
{
    "name":"um_connection1_updated",
    "type":"_VDS_UM_TRANSPORT",
    "resolution":"MULTICAST",
    "address":"224.9.10.11",
    "umMode":"UMP",
    "umpStoreName":"testUMPStore"
}
{
    "name":"um_connection1",
    "type":"_VDS_UM_TRANSPORT",
    "resolution":"MULTICAST",
    "address":"224.9.10.11",
    "umMode":"UMP",
    "umpStoreName":"testUMPStore",
    "resolverPort":"5489",
    "requestTcpPort":"5479",
    "transportTcpPort":"4568",
    "umConfig":"Advanced UM configuration"
}
```

The following is a sample POST body in JSON format for a WebSocket connection:

```
{
    "name":"ws_connection1_updated",
    "type":"_VDS_WS_TRANSPORT",
    "wsMode":"ACKNOWLEDGEMENT",
    "secure": fasle,
    "port":"8956",
    "healthCheckUrl":"status"
}
{
    "name":"ws_connection1_updated",
    "type":"_VDS_WS_TRANSPORT",
    "messagingMode":"STREAMING",
    "wsConnectionType":"WSS",
    "keystorePath":"/data1/crt/keystorepath",
    "keystore_password":"test_keystore_password",
    "truststorePath":"/data1/crt/truststorepath",
    "truststorePassword":"test_truststorePassword",
    "port":"8956",
```

```
    "healthCheckUrl":"status"
}
{
    "name":"ws_connection1_updated",
    "type":"_VDS_WS_TRANSPORT",
    "messagingMode":"STREAMING",
    "wsConnectionType":"WSS",
    "keystorePath":"/data1/crt/keystorepath",
    "keystore_password":"test_keystore_password",
    "truststorePath":"/data1/crt/truststorepath",
    "truststorePassword":"test_truststorePassword",
    "port":"8956",
    "healthCheckUrl":"status",
    "server":"10.25.45.55",
    "advancedConfig":"Advanced WS configuration"
}
```

POST Response in JSON Format

The following is a sample response for the 200 OK code for an Ultra Messaging connection:

```
{
    "id": {connection-id},
    "success": true,
    "name":"um_connection1_updated",
    "type":"_VDS_UM_TRANSPORT",
    "resolution":"MULTICAST",
    "daemon":"127.0.0.1:15380",
    "umMode":"ULB"
}
```

The following is a sample response for the 200 OK code for an WebSocket connection:

```
{
    "id": {connection-id},
    "success": true,
    "name":"ws_connection1_updated",
    "type":"_VDS_WS_TRANSPORT",
    "wsMode":"ACKNOWLEDGEMENT",
    "secure":false,
    "port":"8956",
    "healthCheckUrl":"status"
}
```

The following is the sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": CONNECTION_NOT_FOUND,
            "parameters": [{connection-id}],
            "errorMessage": "Connection with Id: {connection-id} not found"
            }
        ]
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
```

```
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": CONNECTION_ALREADY_EXISTS,
            "parameters": [{error-message}],
            "errorMessage": "Connection name already exists. Exception: {error-message}"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update connection. Exception: {error-message}"
            }
        ]
}
```

Delete Data Connections

Use the POST method to delete a data connection in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a data connection:

```
POST /api/dataflows/{dataflow-id}/connections/{connection-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": CONNECTION_NOT_FOUND,
            "parameters": [{connection-id}],
            "errorMessage": "Connection  with Id: {connection-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update dataflow. Exception : {error-message}."
            }
```

```
        ]
}
```

Links

You can use API calls to perform various operations with links.

Links connect entities in EDS. EDS includes the following types of links:

Link with connection ID

Use this link in one or all of the following scenarios:

- To connect a source service or a transformation on the source service to a target service or a transformation on the target service.
- To connect a source service or a transformation on the source service to an aggregator or a transformation on the aggregator.
- To connect an aggregator or a transformation on the aggregator to a target service or a transformation on the target service.

Link without connection ID

Use this link in one or all of the following scenarios:

- To connect an entity and a transformation on the entity. For example, between an aggregator and a transformation associated with the aggregator.
- To connect two transformations associated with an entity. For example, between two transformations on a source service.

Create Links

Use the POST method to create a link in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a link:

```
POST /api/dataflows/{dataflow-id}/links

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

The following is a sample body for a link with connection ID:

```
{
    "fromId": 1,
    "toIds": [2],
    "connectionId": 3
}
```

The following is a sample body for a link without connection ID:

```
{
    "fromId": 1,
    "toIds": [2],
}
```

POST Response in JSON Format

The following is a sample response for the 201 Created code:

```
{
    "id": {link-id},
    "success": true,
    "fromId": {from-entity-id},
    "toIds": [ {to-entity-id} ],
    "connectionId": {connection-entity-id}
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": ENTITY_NOT_FOUND,
            "parameters": [{from-entity-id}],
            "errorMessage": "Entity with Id: {from-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": LINK_ALREADY_EXISTS,
            "parameters": [{from-entity-id}, {to-entity-id}],
            "errorMessage": "Link already exists between from: {from-entity-id} to: {to-
entity-id}"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{link-id}, {error-message}],
            "errorMessage": "Failed to create link Id: {link-id}.  Exception : {error-
message}."
            }
        ]
}
```

Retrieve Links

Use the GET method to retrieve links based on ID in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a link based on the ID:

```
GET /api/dataflows/{dataflow-id}/links/{link-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {link-id},
    "success": true,
    "fromId": {from-entity-id},
    "toIds": [ {to-entity-id} ],
    "connectionId": {connection-entity-id}
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": LINK_NOT_FOUND,
            "parameters": [{link-id}],
            "errorMessage": "Link with Id: {link-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{link-id}, {error-message}],
            "errorMessage": "Failed to update link Id: {link-id}. Exception : {error-
message}."
            }
        ]
}
```

Retrieve All Links

Use the GET method to retrieve all links in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all links:

```
GET /api/dataflows/{dataflow-id}/links
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is the sample response for the 200 OK code when there are no links in the data flow:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}
```

The following is a sample response when there are two links in the data flow:

```
{
    "total": 2,
    "count": 2,
    "success": true,
    "items": [
       { "id": 46, "fromId": 47, "toIds": [44], "connectionId": 45, "transportTopic":
"FROM_47", "isTransportTopicGenerated": true },
       { "id": 48, "fromId": 43, "toIds": [47] }
    ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all links. Exception : {error-message}."
            }
        ]
}
```

Update Links

Use the POST method to update a link in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a link:

```
POST /api/dataflows/{dataflow-id}/links/{link-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following a sample POST body in JSON format:

```
{
    "fromId": 1,
    "toIds": [ 3 ],
    "connectionId": 3
}
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {link-id},
    "success": true,
    "fromId": {from-entity-id},
```

```
    "toId": {to-entity-id},
    "connectionId": {connection-entity-id}
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": ENTITY_NOT_FOUND,
            "parameters": [{from-entity-id}],
            "errorMessage": "Entity with Id: {from-entity-id} not found"
            }
        ]
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": FROMID_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'fromId' field is missing in request"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": LINK_ALREADY_EXISTS,
            "parameters": [{link-id}],
            "errorMessage": "Link with Id: {link-id} alreday exists"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update source. Exception: {error-message}"
            }
        ]
}
```

Delete Links

Use the POST method to delete a link in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a link:

```
POST /api/dataflows/{dataflow-id}/links/{link-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found."
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": LINK_NOT_FOUND,
            "parameters": [{link-id}],
            "errorMessage": "Link with Id: {link-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{link-id}, {error-message}],
            "errorMessage": "Failed to delete link Id: {link-id}. Exception : {error-
message}."
            }
        ]
}
```

Transformations

You can use API calls to perform various operations with transformations.

Create Transformations

Use the POST method to create a transformation in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a transformation:

```
POST /api/dataflows/{dataflow-id}/transforms

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
    "name": "TestInsertTransform",
    "pluginId": "_VDS_TRX_INSERT_STR",
    "runMode": "WITH_PREVIOUS",
    "config":
    {
        "stringExpression": "#HOSTNAME#DATA"
    },
   "stats":
    [
        {"name": "Events Received", "value": 10301, "type": 102}, {"name": "Bytes
Received", "value": 10300, "type": 102 }, {"name": "Events Generated", "value": 10303,
"type": 102 },
        {"name": "Bytes Sent", "value": 10302, "type": 102 }, {"name": "Time Taken for
Transformation (Milli Sec)", "value": 10304, "type": 102}
    ]
}
```

POST Response in JSON Format

The following is a sample response for the 201 Created code:

```
{
    "id": {transform-entity-id},
    "name": "TestInsertTransform",
    "type": "TRANSFORM",
    "pluginId": "_VDS_TRX_INSERT_STR",
    "runMode": "WITH_PREVIOUS",
    "self":
    {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/transforms/{transform-entity-id}",
        "title": "TestInsertTransform",
        "id": {transform-entity-id}
    },
    "dataflow":
    {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
        "title": "TestDataflow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-07T15:38:48+05:30",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "schedulable": false,
    "config":
    {
        "stringExpression": "#HOSTNAME#DATA"
    },
    "stats":
    [
```

```
        {"name": "Events Received", "value": 10301, "type": 102}, {"name": "Bytes
Received", "value": 10300, "type": 102 }, {"name": "Events Generated", "value": 10303,
"type": 102 },
        {"name": "Bytes Sent", "value": 10302, "type": 102 }, {"name": "Time Taken for
Transformation (Milli Sec)", "value": 10304, "type": 102}
    ]
}
```

The following is a sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to create transform. Exception: {error-message}"
            }
        ]
}
```

Retrieve Transformation by ID

Use the GET method to retrieve transformations based on ID in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a transformation based on the ID:

```
GET /api/dataflows/{dataflow-id}/transforms/{transform-entity-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {transform-entity-id},
    "name": "TestInsertTransform",
```

```
    "type": "TRANSFORM",
    "pluginId": "_VDS_TRX_INSERT_STR",
    "runMode": "WITH_PREVIOUS",
    "self":
    {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/transforms/{transform-entity-id}",
        "title": "TestInsertTransform",
        "id": {transform-entity-id}
    },
    "dataflow":
    {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
        "title": "TestDataflow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-07T15:38:48+05:30",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "schedulable": false,
    "config":
    {
        "stringExpression": "#HOSTNAME#DATA"
    },
    "stats":
    [
        {"name": "Events Received", "value": 10301, "type": 102}, {"name": "Bytes
Received", "value": 10300, "type": 102 }, {"name": "Events Generated", "value": 10303,
"type": 102 },
        {"name": "Bytes Sent", "value": 10302, "type": 102 }, {"name": "Time Taken for
Transformation (Milli Sec)", "value": 10304, "type": 102}
    ]
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": TRANSFORM_ENTITY_NOT_FOUND,
            "parameters": [{transform-entity-id}],
            "errorMessage": "Transform Entity with Id: {transform-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
```

```
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{transform-entity-id}, {error-message}],
            "errorMessage": "Failed to get transform with Id: {transform-entity-id}.
Exception: {error-message}"
            }
        ]
}
```

Retrieve All Transformations

Use the GET method to retrieve all transformations in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all transformations:

```
GET /api/dataflows/{dataflow-id}/transforms

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is the sample response for the 200 OK code when there are no transformations in the data flow:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}
```

The following is the sample response for when there is a transformation in the data flow:

```
{
    "total": 1,
    "count": 1,
    "success": true,
    "items": [
        {
            "id": {transform-entity-id},
            "name": "TestInsertTransform",
            "type": "TRANSFORM",
            "pluginId": "_VDS_TRX_INSERT_STR",
            "runMode": "WITH_PREVIOUS",
            "self": {
                "rel": "self",
                "href": "api/dataflows/{dataflow-id}/transforms/{transform-entity-id}",
                "title": "TestInsertTransform",
                "id": {transform-entity-id}
            },
            "dataflow": {
                "rel": "dataflow",
                "href": "api/dataflows/{dataflow-id}",
                "title": "TestDataflow",
                "id": {dataflow-id}
            },
            "valid": true,
            "createDate": "2015-07-07T15:38:48+05:30",
            "createdBy": "Administrator",
            "predefined": false,
            "deployable": true,
            "deploymentState": "DRAFT",
            "force": false,
            "schedulable": false,
            "config":
```

```
            {
                "stringExpression": "#HOSTNAME#DATA"
            },
            "stats":
            [
                {"name": "Events Received", "value": 10301, "type": 102}, {"name":
"Bytes Received", "value": 10300, "type": 102 }, {"name": "Events Generated", "value":
10303, "type": 102 },
                {"name": "Bytes Sent", "value": 10302, "type": 102 }, {"name": "Time
Taken for Transformation (Milli Sec)", "value": 10304, "type": 102}
            ]
        }
    ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all transforms. Exception: {error-message}"
            }
        ]
}
```

Update Transformation

Use the POST method to update a transformation in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a transformation:

```
POST /api/dataflows/{dataflow-id}/transforms/{transform-entity-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following a sample POST body in JSON format:

```
{
    "name": "TestInsertTransformChanged",
    "type": "TRANSFORM",
    "pluginId": "_VDS_TRX_INSERT_STR",
    "runMode": "WITH_NEXT",
    "config":
    {
        "stringExpression": "#HOSTNAME#DATA#TIMESTAMP"
    },
```

```
   "stats":
    [
        {"name": "Events Received", "value": 10301, "type": 102}, {"name": "Bytes
Received", "value": 10300, "type": 102 }, {"name": "Events Generated", "value": 10303,
"type": 102 },
        {"name": "Bytes Sent", "value": 10302, "type": 102 }, {"name": "Time Taken for
Transformation (Milli Sec)", "value": 10304, "type": 102}
    ]
}
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": {transform-entity-id},
    "name": "TestInsertTransformChanged",
    "type": "TRANSFORM",
    "pluginId": "_VDS_TRX_INSERT_STR",
    "runMode": "WITH_NEXT",
    "self":
    {
        "rel": "self",
        "href": "api/dataflows/{dataflow-id}/transforms/{transform-entity-id}",
        "title": "TestInsertTransformChanged",
        "id": {transform-entity-id}
    },
    "dataflow":
    {
        "rel": "dataflow",
        "href": "api/dataflows/{dataflow-id}",
        "title": "TestDataflow",
        "id": {dataflow-id}
    },
    "success": true,
    "valid": true,
    "createDate": "2015-07-07T15:38:48+05:30",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "schedulable": false,
    "config":
    {
        "stringExpression": "#HOSTNAME#DATA#TIMESTAMP"
    },
    "stats":
    [
        {"name": "Events Received", "value": 10301, "type": 102}, {"name": "Bytes
Received", "value": 10300, "type": 102 }, {"name": "Events Generated", "value": 10303,
"type": 102 },
        {"name": "Bytes Sent", "value": 10302, "type": 102 }, {"name": "Time Taken for
Transformation (Milli Sec)", "value": 10304, "type": 102}
    ]
}
```

The following are a sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
```

```
        ]
}
```

```
{
    "success": false,
    "errors":
        [
            {
            "code": TRANSFORM_ENTITY_NOT_FOUND,
            "parameters": [{transform-entity-id}],
            "errorMessage": "Transform Entity with Id: {transform-entity-id} not found"
            }
        ]
}
```

The following is sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update transform. Exception: {error-message}"
            }
        ]
}
```

Delete Transformations

Use the POST method to delete a transformation in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a transformation:

```
POST /api/dataflows/{dataflow-id}/transforms/{transform-entity-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": DATAFLOW_NOT_FOUND,
            "parameters": [{dataflow-id}],
            "errorMessage": "Dataflow with Id: {dataflow-id} not found"
            }
```

```
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": TRANSFORM_ENTITY_NOT_FOUND,
            "parameters": [{transform-entity-id}],
            "errorMessage": "Transform Entity with Id: {transform-entity-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to delete transform. Exception: {error-message}"
            }
        ]
}
```

Node Groups

You can use API calls to perform various operations with node groups.

Create Node Groups

Use the POST method to create a node group in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a node group:

```
POST /api/nodegroups

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
    "name": "TestNodeGroup",
    "type": "STATIC",
    "nodes": [1, 2, 3]
}
```

POST Response in JSON Format

The following is a sample response for the 201 Created code:

```
{
  "id": 322,
  "name": "NodeGroup2",
  "success": true,
  "type": "STATIC",
  "nodes": [
    {
```

```
      "id": 253
    }
  ]
}
```

The following is a sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": UNSUPPORTED_NODEGROUP_TYPE,
            "parameters": [{invalid-nodegroup-type}],
            "errorMessage": "Unsupported node group type:  {invalid-nodegroup-type}.
Valid node group types (static, dynamic)"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": PATTERN_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'pattern' (regex) field is missing request. It is required
for 'dynamic' Node group"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": NODE_NOT_FOUND,
            "parameters": [{node-id}],
            "errorMessage": "Node not found with Id: {node-id}"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": NODEGROUP_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Node group name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
```

```
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": ""Failed to create Node. Error: {error-message}"
            }
        ]
```

Retrieve Node Groups

Use the GET method to retrieve node groups based on ID in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a node group based on the ID:

```
GET /api/nodegroups/{nodegroup-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following are sample responses for the 200 OK code:

```
{
  "id": 322,
  "name": "NodeGroup2",
  "success": true,
  "type": "STATIC",
  "nodes": [
    {
      "id": 253,
      "name": "nd1"
    }
  ]
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODEGROUP_NOT_FOUND,
            "parameters": [{nodegroup-id}],
            "errorMessage": "Nodegroup with Id: {nodegroup-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get node group. Exception : {error-message}."
            }
        ]
}
```

Retrieve All Node Groups

Use the GET method to retrieve all node groups in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all node groups:

```
GET /api/nodegroups
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is the sample response for the 200 OK code:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}
```

```
{
    "success": true,
    "total": 2,
    "count": 2,
    "items":
    [
        {"id":1, "name":"TestStaticNodeGroup", "type":"static", "nodes":[{"id":1,
"name":"inw00004192"}, {"id":2, "name":"TestNode1"}]},
        {"id":3, "name":"DynamicNodeGroup1", "type":"dynamic", "pattern":"node*",
"nodes":[{"id":9, "name":"node1"}]}
    ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all node groups. Exception : {error-message}."
            }
        ]
}
```

Update Node Groups

Use the POST method to update a node group in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a node group:

```
POST /api/nodegroups/{nodegroup-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following is a sample POST body in JSON format:

```
{
    "name": "TestNodeGroupChnaed",
```

```
    "type": "STATIC",
    "nodes": [3, 8]
}
```

POST Response in JSON Format

The following are sample responses for the 200 OK code:

```
{
    "id": {nodegroup-id},
    "name": {nodegroup-name},
    "success": true,
    "type": "STATIC",
    "nodes": [{"id": {node-id}, "name": {node-name}}]
}

{
    "id": {nodegroup-id},
    "name": {nodegroup-name},
    "success": true,
    "type": "dynamic",
    "pattern": {node-pattern-regex},
    "nodes": [{"id": {node-id}, "name": {node-name}}]
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODEGROUP_NOT_FOUND,
            "parameters": [{nodegroup-id}],
            "errorMessage": "Nodegroup with Id: {nodegroup-id} not found"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": NODE_NOT_FOUND,
            "parameters": [{node-id}],
            "errorMessage": "Node with Id: {node-id} not found"
            }
        ]
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": CANNOT_CHANGE_NODEGROUP_TYPE,
            "parameters": null,
```

```
            "errorMessage": "Cannot change nodegroup type"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update node group. Exception : {error-message}."
            }
        ]
}
```

Delete Node Groups

Use the POST method to delete a node group in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a node group:

```
POST /api/nodegroups/{nodegroup-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODEGROUP_NOT_FOUND,,
            "parameters": [{nodegroup-id}],
            "errorMessage": "Nodegroup with Id: {nodegroup-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to delete node group. Exception : {error-message}."
            }
        ]
}
```

Nodes

You can use API calls to perform various operations with nodes.

Create Nodes

Use the POST method to create a node in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a node:

```
POST /api/nodes

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
    "nodes":
        [
            {"name": "MyNode1"},
            {"name": "MyNode2", "nodeGroups": [1]},
        ]
}
```

POST Response in JSON Format

The following is a sample response for the 201 Created code:

```
{
    "total": 1,
    "count": 1,
    "success": true,
    "items":
    [
        {
            "id": {node-id},
            "name": {node-name},
            "nodeGroups": [{"id": {nodegroup-id}, "name": {nodegroup-name}, "type":
"static"}]
        }
    ]
}
```

```
{
    "total": 1,
    "count": 1,
    "success": true,
    "items":
    [
        {
            "id": {node-id},
            "name": {node-name},
            "nodeGroups": [{"id": {nodegroup-id}, "name": {nodegroup-name}, "type":
"dynamic", "pattern": {node-pattern-regex}}]
        }
    ]
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
```

```
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": NODE_NAME_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Node name already exists."
            }
        ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODEGROUP_NOT_FOUND,
            "parameters": [{nodegroup-id}],
            "errorMessage": "Nodegroup not found with Id: {nodegroup-id}"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to create node. Exception: {error-message}"
            }
        ]
}
```

Retrieve Nodes

Use the GET method to retrieve nodes in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a node:

```
GET /api/nodes/{node-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following are sample responses for the 200 OK code:

```
{
    "id": {node-id},
    "name": {node-name},
    "success": true,
    "nodeGroups": [{"id": {nodegroup-id}, "name": {nodegroup-name}, "type": "static"}]
}

{
    "id": {node-id},
    "name": {node-name},
```

```
    "success": true,
    "nodeGroups":
        [
            {"id": {nodegroup-id}, "name": {nodegroup-name}, "type": "dynamic",
"pattern": {node-pattern-regex}},
            {"id": {nodegroup-id}, "name": {nodegroup-name}, "type": "dynamic",
"pattern": {node-pattern-regex}}
        ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODE_NOT_FOUND,
            "parameters": [{node-id}],
            "errorMessage": "Node with Id: {node-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get node. Exception : {error-message}."
            }
        ]
}
```

Retrieve All Nodes

Use the GET method to retrieve all nodes in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all nodes:

```
GET /api/nodes

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following are sample responses for the 200 OK code:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}

{
    "success": true,
    "total":3,
    "count":3,
    "items":[
        {"id": {node-id}, "name": {node-name}, "nodeGroups":[{"id": {nodegroup-id},
"name": {nodegroup-name}, "type":"static"},
        {"id": {node-id}, "name": {node-name}, "nodeGroups":[{"id": {nodegroup-id},
```

```
"name": {nodegroup-name}, "type":"dynamic", "pattern": {node-pattern-regex}}]},
        {"id": {node-id}, "name": {node-name}, "nodeGroups":[]}
       ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all nodes. Exception : {error-message}."
            }
        ]
}
```

Update Nodes

Use the POST method to update a node in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a node:

```
POST /api/nodes/{node-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following is a sample POST body in JSON format:

```
{
    "nodes":
        [
            {"id": 1, "name": "MyNode1Chnaged"},
            {"id": 2, "name": "MyNode2Changed", "nodeGroups": [5, 8]},
        ]
}
```

POST Response in JSON Format

The following are sample responses for the 200 OK code:

```
{
    "total": 1,
    "count": 1,
    "success": true,
    "items":
    [
        {
            "id": {node-id},
            "name": {node-name},
            "nodeGroups": [{"id": {nodegroup-id}, "name": {nodegroup-name}, "type":
"dynamic", "pattern": {node-pattern-regex}}]
        }
    ]
}

{
    "total": 2,
    "count": 2,
    "success": true,
    "items":
    [
```

```
        {
            "id": {node-id},
            "name": {node-name},
            "nodeGroups": [{"id": {nodegroup-id}, "name": {nodegroup-name}, "type":
"dynamic", "pattern": {node-pattern-regex}}]
        },
        {
            "id": {node-id},
            "name": {node-name},
            "nodeGroups": [{"id": {nodegroup-id}, "name": {nodegroup-name}, "type":
"static"}]
        }
    ]
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODE_NOT_FOUND,
            "parameters": [{node-id}],
            "errorMessage": "Node with Id: {node-id} not found"
            }
        ]
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NAME_FIELD_MISSING,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}
```

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODEGROUP_NOT_FOUND,
            "parameters": [{nodegroup-id}],
            "errorMessage": "Nodegroup not found with Id: {nodegroup-id}"
            }
        ]
}
```

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODE_NAME_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Node name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
```

```
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update Node. Exception: {error-message}"
            }
        ]
}
```

Delete Nodes

Use the POST method to delete a node in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a node:

```
POST /api/nodes/{node-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following is a sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": NODE_NOT_FOUND,
            "parameters": [{node-id}],
            "errorMessage": "Node with Id: {node-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to delete node. Exception : {error-message}."
            }
        ]
}
```

Parameters

You can use API calls to perform various operations with parameters.

Create Parameters

Use the POST method to create a parameter in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates a parameter:

```
POST /api/parameters

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
    "parameters":
        [
            {"key": "TestKey", "value": "TestValue"},
            {"key": "Key2", "value": "Value2", "secure": true},
        ]
}
```

POST Response in JSON Format

The following are sample responses for the 201 Created code:

```
{
    "total": 1,
    "count": 1,
    "success": true,
    "items":
    [
        {"parameterId": {parameter-id}, "key": {parameter-key}, "success": true,
"value": {parameter-value-masked}, "secure": false, "timestamp": {timestamp}} }
    ]
}

{
    "total": 2,
    "count": 2,
    "success": true,
    "items":
    [
        {"parameterId": {parameter-id}, "key": {parameter-key}, "success": true,
"value": {parameter-value-masked}, "secure": true, "timestamp": {timestamp}},
        {"parameterId": {parameter-id}, "key": {parameter-key}, "success": true,
"value": {parameter-value}, "secure": false, "timestamp": {timestamp}}
    ]
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": PARAMETER_KEY_FIELD_NOT_FOUND,
            "parameters": null,
            "errorMessage": "'key' field is missing in request"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": PARAMETER_NAME_ALREADY_EXISTS,
            "parameters": null,
```

```
            "errorMessage": "Parameter name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to create parameter. Exception: {error-message}"
            }
        ]
}
```

Retrieve Parameters

Use the GET method to retrieve parameters in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a node:

```
GET /api/parameters/{parameter-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following are sample responses for the 200 OK code:

```
{
    "parameterId": {parameter-id},
    "key": {parameter-key},
    "success": true,
    "value": {parameter-value-masked},
    "secure": false,
    "timestamp": {timestamp}
}

{
    "parameterId": {parameter-id},
    "key": {parameter-key},
    "success": false,
    "value": {parameter-value},
    "secure": true,
    "timestamp": {timestamp}
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": PARAMETER_NOT_FOUND,
            "parameters": [{parameter-id}],
            "errorMessage": "Parameter with Id: {parameter-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get parameter. Exception : {error-message}."
            }
        ]
}
```

Retrieve All Parameters

Use the GET method to retrieve all parameters in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all parameters:

```
GET /api/parameters
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following are the sample responses for the 200 OK code:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}

{
    "success": true,
    "total":3,
    "count":3,
    "items":
    [
        {"key": "key1", "value": "*****", "secure": true, "timestamp": 1435911720961,
"parameterId": 1},
        {"key": "key2", "value": "value2", "secure": false, "timestamp": 1435911720924,
"parameterId": 2},
        {"key": "TestKey", "value": "TestValue", "secure": false, "timestamp":
1435911720785, "parameterId": 3}
    ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all parameters. Exception : {error-message}."
            }
        ]
}
```

Update Parameters

Use the POST method to update a parameter in VDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates a parameter:

```
POST /api/parameters

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

The following a sample POST body in JSON format:

```
{
    "parameters":
        [
            {"parameterId": 1, "key": "TestKeyChnaged", "value": "TestValue"},
            {"parameterId": 2, "key": "Key2Changed", "value": "Value2Updated", "secure":
true},
        ]
```

POST Response in JSON Format

The following are sample responses for the 200 OK code:

```
{
    "total": 1,
    "count": 1,
    "success": true,
    "items":
    [
        {"parameterId": {parameter-id}, "key": {parameter-key}, "success": true,
"value": {parameter-value}, "secure": false, "timestamp": {timestamp}} }
    ]
}
```

```
{
    "total": 2,
    "count": 2,
    "success": true,
    "items":
    [
        {"parameterId": {parameter-id}, "key": {parameter-key}, "success": true,
"value": {parameter-value-masked}, "secure": true, "timestamp": {timestamp}},
        {"parameterId": {parameter-id}, "key": {parameter-key}, "success": true,
"value": {parameter-value}, "secure": false, "timestamp": {timestamp}}
    ]
}
```

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": PARAMETER_NOT_FOUND,
            "parameters": [{parameter-id}],
            "errorMessage": "Parameter with Id: {parameter-id} not found"
            }
        ]
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
```

```
    "errors":
        [
            {
            "code": PARAMETER_ID_FIELD_NOT_FOUND,
            "parameters": null,
            "errorMessage": "'parameterId' field is missing in request"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": PARAMETER_KEY_FIELD_NOT_FOUND,
            "parameters": null,
            "errorMessage": "'key' field is missing in request"
            }
        ]
}

{
    "success": false,
    "errors":
        [
            {
            "code": PARAMETER_NAME_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Parameter name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update parameter. Exception : {error-message}"
            }
        ]
}
```

Delete Parameters

Use the POST method to delete a parameter in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes a parameter:

```
POST /api/parameters/{parameter-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following are sample responses for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
```

```
            {
            "code": PARAMETER_NOT_FOUND,
            "parameters": [{parameter-id}],
            "errorMessage": "Parameter with Id: {parameter-id} not found"
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{parameter-id}, {error-message}],
            "errorMessage": "Failed to delete parameter with Id:{parameter-id}.
Exception: {error-message}"
            }
        ]
}
```

Plug-ins

You can use API calls to perform various operations with plug-ins.

Retrieve Plugins

Use the GET method to retrieve plugins in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves a plugin:

```
GET /api/plugins/{plugin-id}
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "name": "Syslog TCP",
    "success": true,
    "pluginId": "_VDS_SRC_SYSLOG_TCP",
    "zipName": "C:\\Informatica\\vds\\6.5\\admind\\plugins\\infa\\syslog-tcp-2.3.0.zip",
    "fieldsInfo": {"helpKey":"syslogTCP_source_properties","sourceSystemProperties":
[{"textControl":{"integerTextField":{"default":8192,"minValue":1},"description":"Size of
the source data (bytes) to be sent in an event. An event consists of one or more
complete records.","name":"eventSize","mandatory":true,"displayName":"Maximum Event
Size"},"seqNum":4},{"seqNum":6,"checkBoxControl":{"default":false,"description":"Persist
the data from source locally so that source can continue processing even if the target
is down.","name":"persistEvents","displayName":"Persist Data"}},{"seqNum":
7,"textAreaControl":{"default":"# --\\n# The interval in milliseconds after which the
writes \\n# are flushed to disk.\\n# --\\n#flushInterval=5000\\n\\n# --\\n# The maximum
number of db files to keep\\n# --\\n#maxDataFiles=10\\n\\n# --\\n# Size of individual db
file in bytes after which the files are rolled over\\n# --\\n#maxDataFileSize=1073741824\
\n\\n# --\\n# The max size of unsent data in bytes. If 0, it is unbounded\\n# --\
\n#maxQueueSizeInBytes=0\\n\\n# --\\n# The batch size in bytes up to which the data is
buffered before writing to the disk.\\n# --\\n#batchSizeInBytes=262144\\n\
\n","description":"Advanced options; only valid if persist data option is
selected","name":"persistOptions","displayName":"Persistence
```

```
options"}}],"displayName":"Syslog TCP","version":"2.3.0_1435821164","fields":
[{"textControl":{"description":"Name of the entity. Maximum number of characters is
32.","name":"name","mandatory":true,"stringTextField":{"maxLength":32,"pattern":"^([_a-
zA-Z][_a-zA-Z0-9 ]*)$","secure":false},"displayName":"Entity Name"},"seqNum":1},
{"textControl":{"integerTextField":{"minValue":1,"maxValue":
65536},"name":"portTCP","mandatory":true,"displayName":"Port"},"seqNum":2},{"seqNum":
3,"listControl":{"default":0,"items":[{"id":
0,"displayName":"LF"}],"name":"delimiter","displayName":"Delimiter"}}]},
    "runTime":
{"helpKey":"syslogTCP_source_properties","pluginClass":"com.informatica.binge.sources.sys
log_tcp.SyslogTcpReader","pluginId":"_VDS_SRC_SYSLOG_TCP","name":"Syslog
TCP","pluginJar":"syslog-tcp.jar","version":"2.3.0_1435821164"},
    "statistics": {"statistic":[{"id":"1","displayName":"Events
Dropped","type":"CUMULATIVE"},{"id":"2","displayName":"Concurrent
Connections","type":"CUMULATIVE"},{"id":"3","displayName":"Maximum Concurrent
Clients","type":"CUMULATIVE"}]},
    "type": "SOURCE"
}
```


The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": PLUGIN_NOT_FOUND,
            "parameters": [{plugin-id}],
            "errorMessage": "Plugin with id {plugin-id} not found."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get plugin. Exception : {error-message}."
            }
        ]
}
```

Retrieve All Plugins

Use the GET method to retrieve all plugins in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all plugins:

```
GET /api/plugins
Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "success": true,
    "total": 35,
    "count": 35,
    "items":
    [
        { "pluginId": "_VDS_SRC_FLAT_FILE",     "name": "File",
```

```
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_HTTP",          "name": "HTTP(S)",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_JMS",           "name": "JMS",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_MQTT",          "name": "MQTT",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_STATIC_FILE",   "name": "Static File",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_SYSLOG_TCP",    "name": "Syslog TCP",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_SYSLOG_UDP",    "name": "Syslog UDP",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_SYSLOG_UDS",    "name": "Syslog UDS",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_TCP",           "name": "TCP",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_UDP",           "name": "UDP",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_UM",            "name": "Ultra Messaging",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_OPC_DA",        "name": "OPC DA",
"type": "SOURCE" },
        { "pluginId": "_VDS_SRC_WS",            "name": "WebSocket(S)",
"type": "SOURCE" },
        { "pluginId": "_VDS_TGT_CASSANDRA",     "name": "Cassandra",
"type": "TARGET" },
        { "pluginId": "_VDS_TGT_FILE",          "name": "File",
"type": "TARGET" },
        { "pluginId": "_VDS_TGT_HDFS",          "name": "HDFS",
"type": "TARGET" },
        { "pluginId": "_VDS_TGT_HTTP",          "name": "HTTP(S)",
"type": "TARGET" },
        { "pluginId": "_VDS_TGT_JMS",           "name": "JMS",
"type": "TARGET" },
        { "pluginId": "_VDS_TGT_KINESIS",       "name": "Kinesis",
"type": "TARGET" },
        { "pluginId": "_VDS_TGT_WS",            "name": "WebSocket(S)",
"type": "TARGET" },
        { "pluginId": "_VDS_TGT_EVENTHUB",      "name": "EventHub",
"type": "TARGET" },
        { "pluginId": "_VDS_TRX_COMPRESS",      "name": "Compress Data",
"type": "TRANSFORM" },
        { "pluginId": "_VDS_TRX_DECOMPRESS",    "name": "Decompress Data",
"type": "TRANSFORM" },
        { "pluginId": "_VDS_TRX_INSERT_STR",    "name": "Insert String",
"type": "TRANSFORM" },
        { "pluginId": "_VDS_TRX_JS",            "name": "JavaScript",
"type": "TRANSFORM" },
        { "pluginId": "_VDS_TRX_REGEX",         "name": "Regex Filter",
"type": "TRANSFORM" },
        { "pluginId": "_VDS_TRX_UDPARSER",      "name": "Unstructured Data Parser",
"type": "TRANSFORM" },
        { "pluginId": "_VDS_AGGREGATOR",        "name": "Aggregator",
"type": "AGGREGATOR" },
        { "pluginId": "_VDS_UM_TRANSPORT",      "name": "Ultra Messaging",
"type": "TRANSPORT" },
        { "pluginId": "_VDS_WS_TRANSPORT",      "name": "WebSocket(S)",
"type": "TRANSPORT" }
    ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
```

```
                "errorMessage": "Failed to get all plugins. Exception : {error-message}."
                }
            ]
}
```

Aggregators

You can use API calls to perform various operations with aggregators.

Create Aggregators

Use the POST method to create an aggregator in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request creates an aggregator:

```
POST /api/dataflows

Content-Type: application/json
Accept: application/json
```

POST Body in JSON Format

```
{
    "config": {
        "description": "",
        "eventQueueSize": 100000,
        "topicName": "",
        "topicDescription": "",
        "topicExpiresIn": "600000",
        "topicResponderAccess": "NAMED_PROPERTIES"
    },
    "stats": [ { "name": "Bytes Received", "value": 10501, "type": 101 },
               { "name": "Events Received", "value": 10503, "type": 101 },
               { "name": "Receive Rate(Per Sec)", "value": 10504, "type": 101 },
               { "name": "Bytes Sent", "value": 10001, "type": 101 },
               { "name": "Events Sent", "value": 10003, "type": 101 },
               { "name": "Send Rate(Per Sec)", "value": 10005, "type": 101 },
               { "name": "Events Reassigned", "value": 10505, "type": 101 },
               { "name": "Events to be Processed", "value": 10601, "type": 101 }],
    "name": "aggregator2",
    "pluginId": "_VDS_AGGREGATOR",
    "topics": [],
    "topicProperties": []
}
```

POST Response in JSON Format

The following is a sample response for the 201 Created code:

```
{
    "id": 4,
    "name": "aggregator2",
    "self": {
        "rel": "self",
        "href": "api/dataflows/1/aggregator/4",
        "title": "aggregator2",
        "id": 4
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/1",
        "title": "Data flow",
        "id": 1
```

```
    },
    "type": "AGGREGATOR",
    "valid": true,
    "createDate": "2015-07-22T02:17:02-07:00",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": true,
    "deploymentState": "DRAFT",
    "force": false,
    "pluginId": "_VDS_AGGREGATOR",
    "schedulable": false,
    "config": {
        "topicExpiresIn": "600000",
        "topicDescription": "",
        "description": "",
        "topicResponderAccess": "NAMED_PROPERTIES",
        "eventQueueSize": "100000",
        "topicName": "",
    },
    "stats": [ { "name": "Bytes Received", "value": 10501, "type": 101 },
                { "name": "Events Received", "value": 10503, "type": 101 },
                { "name": "Receive Rate(Per Sec)", "value": 10504, "type": 101 },
                { "name": "Bytes Sent", "value": 10001, "type": 101 },
                { "name": "Events Sent", "value": 10003, "type": 101 },
                { "name": "Send Rate(Per Sec)", "value": 10005, "type": 101 },
                { "name": "Events Reassigned", "value": 10505, "type": 101 },
                { "name": "Events to be Processed", "value": 10601, "type": 101 }],
    "topics": [],
    "topicProperties": []
}
}
```

The following are sample responses for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": AGGREGATOR_NAME_FIELD_NOT_FOUND,
            "parameters": null,
            "errorMessage": "'name' field is missing in request"
            }
        ]
}


{
    "success": false,
    "errors":
        [
            {
            "code": AGGREGATOR_NAME_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Aggregator name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to create aggregator. Exception: {error-message}"
            }
        ]
}
```

Retrieve Aggregators

Use the GET method to retrieve aggregators in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves an aggregator:

```
GET /api/dataflows/{dataflowId:\\d+}/aggregators/{aggregator-id}

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": 3,
    "name": "Aggregator1",
    "self": {
        "rel": "self",
        "href": "api/dataflows/1/aggregator/3",
        "title": "Aggregator1",
        "id": 3
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/1",
        "title": "Data flow",
        "id": 1
    },
    "type": "AGGREGATOR",
    "valid": true,
    "createDate": "2015-07-21T23:47:20-07:00",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": false,
    "force": false,
    "pluginId": "_VDS_AGGREGATOR",
    "schedulable": false,
    "config": {
        "topicExpiresIn": "600000",
        "topicDescription": "",
        "description": "",
        "topicResponderAccess": "NAMED_PROPERTIES",
        "topicName": "",
        "eventQueueSize": "100000"
    },
    "stats": [
                { "name": "Bytes Received", "value": 10501, "type": 101 },
                { "name": "Events Received", "value": 10503, "type": 101 },
                { "name": "Receive Rate(Per Sec)", "value": 10504, "type": 101 },
                { "name": "Bytes Sent", "value": 10001, "type": 101 },
                { "name": "Events Sent", "value": 10003, "type": 101 },
                { "name": "Send Rate(Per Sec)", "value": 10005, "type": 101 },
                { "name": "Events Reassigned", "value": 10505, "type": 101 },
                { "name": "Events to be Processed", "value": 10601, "type": 101 }
    ],
    "topics": [],
    "topicProperties": []
}
```

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": AGGREGATOR_NOT_FOUND,
```

```
            "parameters": [{aggregator-id}],
            "errorMessage": "Aggregator with Id: {aggregator-id} not found."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get aggregator. Exception : {error-message}."
            }
        ]
}
```

Retrieve All Aggregators

Use the GET method to retrieve all aggregators in EDS.

GET Request in JSON Format

Generate a GET request. For example, the following GET request retrieves all aggregators:

```
GET /api/dataflows/{dataflowId:\\d+}/aggregators

Content-Type: application/json
Accept: application/json
```

GET Response in JSON Format

The following are sample responses for the 200 OK code:

```
{
    "success": true,
    "total": 0,
    "count": 0,
    "items": []
}

{
    "success": true,
    "total": 1,
    "count": 1,
    "items": [
        {
            "id": 3,
            "name": "Aggregator1",
            "self": {
                "rel": "self",
                "href": "api/dataflows/1/aggregator/3",
                "title": "Aggregator1",
                "id": 3
            },
            "dataflow": {
                "rel": "dataflow",
                "href": "api/dataflows/1",
                "title": "Data flow",
                "id": 1
            },
            "type": "AGGREGATOR",
            "valid": true,
            "createDate": "2015-07-21T23:47:20-07:00",
            "createdBy": "Administrator",
            "predefined": false,
            "deployable": false,
```

```
            "force": false,
            "pluginId": "_VDS_AGGREGATOR",
            "schedulable": false,
            "config": {
                "topicExpiresIn": "600000",
                "topicDescription": "",
                "description": "",
                "topicResponderAccess": "NAMED_PROPERTIES",
                "topicName": "",
                "eventQueueSize": "100000"
            },
            "stats": [
                { "name": "Bytes Received", "value": 10501, "type": 101 },
                { "name": "Events Received", "value": 10503, "type": 101 },
                { "name": "Receive Rate(Per Sec)", "value": 10504, "type": 101 },
                { "name": "Bytes Sent", "value": 10001, "type": 101 },
                { "name": "Events Sent", "value": 10003, "type": 101 },
                { "name": "Send Rate(Per Sec)", "value": 10005, "type": 101 },
                { "name": "Events Reassigned", "value": 10505, "type": 101 },
                { "name": "Events to be Processed", "value": 10601, "type": 101 }
            ],
            "topics": [],
            "topicProperties": []
        }
    ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to get all aggregator. Exception : {error-message}."
            }
        ]
}
```

Update Aggregator

Use the POST method to update an aggregator in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request updates an aggregator:

```
POST /api/dataflows/{dataflowId:\\d+}/aggregators

Content-Type: application/json
Accept: application/json
x-http-method-override: PUT
```

POST Body in JSON Format

```
{
    "config": {
        "description": "",
        "eventQueueSize": "100000",
        "topicName": "",
        "topicDescription": "",
        "topicExpiresIn": "",
        "topicResponderAccess": "NAMED_PROPERTIES"
    },
    "stats": [{ "name": "Bytes Received", "value": 10501, "type": 101 },
                { "name": "Events Received", "value": 10503, "type": 101 },
                { "name": "Receive Rate(Per Sec)", "value": 10504, "type": 101 },
```

```
                { "name": "Bytes Sent", "value": 10001, "type": 101 },
                { "name": "Events Sent", "value": 10003, "type": 101 },
                { "name": "Send Rate(Per Sec)", "value": 10005, "type": 101 },
                { "name": "Events Reassigned", "value": 10505, "type": 101 },
                { "name": "Events to be Processed", "value": 10601, "type": 101 }],
    "name": "aggregator_updated",
    "pluginId": "_VDS_AGGREGATOR",
    "topics": [],
    "topicProperties": []
}
```

POST Response in JSON Format

The following is a sample response for the 200 OK code:

```
{
    "id": 4,
    "name": "aggregator_updated",
    "self": {
        "rel": "self",
        "href": "api/dataflows/1/aggregator/4",
        "title": "aggregator_updated",
        "id": 4
    },
    "dataflow": {
        "rel": "dataflow",
        "href": "api/dataflows/1",
        "title": "Data flow",
        "id": 1
    },
    "type": "AGGREGATOR",
    "valid": true,
    "createDate": "2015-07-22T02:17:02-07:00",
    "createdBy": "Administrator",
    "predefined": false,
    "deployable": false,
    "force": false,
    "pluginId": "_VDS_AGGREGATOR",
    "schedulable": false,
    "config": {
        "topicExpiresIn": "",
        "topicDescription": "",
        "description": "",
        "topicResponderAccess": "NAMED_PROPERTIES",
        "topicName": "",
        "eventQueueSize": "100000"
    },
    "stats": [{ "name": "Bytes Received", "value": 10501, "type": 101 },
                { "name": "Events Received", "value": 10503, "type": 101 },
                { "name": "Receive Rate(Per Sec)", "value": 10504, "type": 101 },
                { "name": "Bytes Sent", "value": 10001, "type": 101 },
                { "name": "Events Sent", "value": 10003, "type": 101 },
                { "name": "Send Rate(Per Sec)", "value": 10005, "type": 101 },
                { "name": "Events Reassigned", "value": 10505, "type": 101 },
                { "name": "Events to be Processed", "value": 10601, "type": 101 }],
    "topics": [],
    "topicProperties": []
}
```

The following is the sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": AGGREGATOR_NOT_FOUND,
            "parameters": [{aggregator-id}],
            "errorMessage": "Aggregator with Id: {aggregator-id} not found"
            }
        ]
}
```

The following is a sample response for the 400 Bad Request code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": AGGREGATOR_NAME_ALREADY_EXISTS,
            "parameters": null,
            "errorMessage": "Aggregator name already exists."
            }
        ]
}
```

The following is a sample response for the 500 Internal Server Error code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{error-message}],
            "errorMessage": "Failed to update aggregator. Exception : {error-message}"
            }
        ]
}
```

Delete Aggregators

Use the POST method to delete an aggregator in EDS.

POST Request in JSON Format

Generate a POST request. For example, the following POST request deletes an aggregator:

```
POST /api/dataflows/{dataflowId:\\d+}/aggregators/{aggregator-id}

Content-Type: application/json
Accept: application/json
x-http-method-override: DELETE
```

POST Response in JSON Format

The following is a sample response for the 404 Not Found code:

```
{
    "success": false,
    "errors":
        [
            {
            "code": AGGREGATOR_NOT_FOUND,
            "parameters": [{aggregator-id}],
            "errorMessage": "Aggregator with Id: {aggregator-id} not found"
            }
        ]
}
500 Internal Server Error

{
    "success": false,
    "errors":
        [
            {
            "code": INTERNAL_SERVER_ERROR,
            "parameters": [{aggregator-id}, {error-message}],
            "errorMessage": "Failed to delete aggregator with Id:{aggregator-id}.
Exception: {error-message}"
            }
        ]
}
```

Authentication

Perform user authentication in the Administrator tool before you perform REST calls.

1. Send a POST request to the following URL:

   `http://<hostname>:<port>/administrator/Login.do`

   Specify the username and password in x-www-form-urlencoded format.

   EDS creates a cookie that it uses to authenticate your username in the Administrator tool.
2. After EDS creates the cookie, use the following base URL to access the REST APIs:

   `http://<hostname>:<port>/administrator/api`

CHAPTER 7

Glossary

Administrator Daemon

Daemon process that facilitates the creation, management, deployment, and undeployment of data flows through the Administrator tool. The Administrator Daemon also aggregates statistics and state information from Edge Data Streaming Nodes in the deployment, and send the information to the Administrator tool.

data flow

Defines the path of data from source services to target services through zero or more transformations. You can create, design, deploy, and undeploy data flows in the Administrator tool. Data flows can be simple data flows, such as one-to-one, or complex data flows, such as, one-to-many, many-to-one, and many-to-many.

EDS Node

An EDS Node is a Java program within which source services and target services run. You can run multiple source services and target services on a single node. You can also configure multiple nodes to run on a host machine.

Informatica Administrator

Informatica Administrator (Administrator tool) is a web application that you can use to manage, monitor, deploy, and undeploy data flows.

receiver type ID

A 32-bit value that uniquely identifies the Ultra Messaging receiver.

source service

A EDS Node contains one or more specialized threads that work together to transfer data from an application host to a target data store or data engine. Source services are threads that consume and publish events generated by a source application. Source services publish data in the form of Ultra Messaging or WebSocket messages.

target service

Target services are the threads that subscribe to data published by source services and write the data to the target. Target services run on hosts that have access to the target.

topic resolution domain

The domain of UDP multicast or unicast connectivity that allows Ultra Messaging topic resolution to occur. Topic resolution enables receivers to discover sources.

unicast topic resolution daemon (LBMRD)

A daemon process that performs the same topic resolution activities as those performed by multicast topic resolution. By default, Ultra Messaging expects multicast connectivity between sources and targets. When only unicast connectivity is available, you must run one or more unicast topic resolution daemons (LBMRD).

INDEX

R

T

U

V