PUBLIC
SAP Cloud Platform Integration for Processes
2020-10-26

Developing Adapters

THE BEST RUN SAP

Content

1 Adapter Development Prerequisites

You have read SAP Cloud Platform Integration Developers' guide and Operations guide so that you have a better understanding of integration flows and connectivity options of SAP Cloud Platform Integration. To extend the connectivity options of SAP Cloud Platform Integration, you can develop new adapter tailored along specific requirements.

You can contribute an Apache Camel component as SAP Cloud Platform Integration adapter. To integrate camel component into SAP Cloud Platform Integration framework, you should specify the detailed set of adapter attributes as component metadata. Component metadata defines the adapter attributes, dependencies between them and the structure of configuration interface for a specific adapter, which is used for user interface generation.

Following are the guidelines that you need to understand and follow while developing camel component:

- **Camel Component Scheme** - Make sure that you use a unique camel component scheme to avoid runtime collisions with other registered schemes.For example, the camel component scheme can be sap-dropbox.
- **Secure Programming** - Secure parameters must be stored in runtime secure store and accessed through alias configured in adapter.
- **Security Aspects** -
  - Always scan for virus in the Adapter workspace / project before deployment.
  - The Adapter design time content bundle should not contain any confidential information (For example: hard-coded secrets, passwords, users or documents).
  - Perform fortify and java static code checks.
- **Logging** - Use Slf4j logging
- **API Consumption**
  - To access platform specific features, you can use public APIs which are available with the Adapter Development Kit (ADK). Public APIs for the ADK are available in SAP Cloud Platform Integration tools page.
  - JavaDoc containing public API can be found at: https://uacp.hana.ondemand.com/http.svc/rc/PRODUCTION/saphci_javadoc_adk/1/en-EN/index.html.
  - To consume the APIs you need to use the code snippet `ITAiFactory.getApi(apiType, context);` wherein, apiType is the class of the requested API interface. The camel context can be null if it is not required for a particular API. For Example: For accessing KeystoreService, use ITApiFactory.getApi(KeystoreService.class, context)

> **i Note**
>
> The Java standard libraries of Java 8 can be used.
>
> Cloud Integration supports the XML Document Object Model (DOM) to process XML documents.

- **Manifest.MF + Whitelisting** - Only ADK APIs, Camel core and Logging are accesible at runtime. However, if required you can import packages for dependent libraries bundled alongwith project.
- You can use the following code snippet, if you want to read a resource in integration flow project at runtime.

> **Sample Code**

```
        InputStream is =
ResourceHelper.resolveMandatoryResourceAsInputStream(getCamelContext().getC
lassResolver(), "//resources/parameters.propdef");
```

2 Adapter Development Process

You develop new SAP Cloud Platform Integration adapters to extend the connectivity of SAP Cloud Platform Integration with remote systems.

SAP Cloud Platform Integration provides a set of adapters to meet standard connectivity needs. For example, the SOAP adapter allows you to exchange SOAP messages through SAP Cloud Platform Integration, or the SFTP adapter supports the exchange of files with SFTP servers. These adapters can be configured in integration flow to enable connectivity between sender and reciever systems.

2.1 Create OSGi Bundle

OSGi bundles are normal jar components with extra manifest headers, which you use to develop adapters.

Prerequisites

- You have the required jar file.
- You have setup the eclipse with plug-in development enabled.

Context

To integrate camel component into SAP Cloud Platform Integration framework, camel component and its dependencies should be OSGi bundle. To convert the jar file to OSGi bundle execute the following steps:

Procedure

1. In Eclipse, go to *File* > *New* > *Project*.
2. In the *New Project* wizard, search for *Plug-in Development* and select *Plug-in from Existing JAR Archives*.
3. Choose *Next*.
4. If you have the JAR in the current workspace, then choose *Add* and select the JAR file from the JAR selection dialog.
5. If you do not have the JAR in your current workspace, then choose *Add External* and select the required JAR file from your local system.
6. Choose *Next*.

7. In the *Plug-in Project Properties*dialog, specify a name in the *Project name* field.
8. In the *Execution Environment* drop down list box, select any java version that is less than or equal to JavaSE-1.7.
9. In the *Target Platform*, select *an OSGi framework* option.
10. Choose *Finish*.
11. Select the MANIFEST.MF file of your project, and go to *Runtime* tab.
12. Choose *Add*.
13. Select the required export packages.
14. Choose *OK* and save the MANIFEST.MF file.
15. Right Click on the project and choose *Export*.
16. In the *Export* wizard search for *plug-in Development* and choose *Deployable plug-ins and fragments*.
17. Choose *Next*.
18. Choose *Browse* and select the directory where you want the plug-in to be generated.
19. Choose *Finish*.

Results

In the directory, you can see the plug-ins folder with the OSGi bundle of jar file.

2.2 Develop Adapters

You can develop custom adapters using Adapter SDK in SAP Cloud Platform Integration.

Prerequisites

- You are aware of the technical details of component/domain that you have to connect with SAP Cloud Platform Integration through the newly developed adapter type.
- You have Java 7 version.
- You are familiar with using Apache Camel and Java.
- You have the Camel component and its dependencies as OSGi bundles.
- You have the camel core runtime version 2.17.4.
- You have the simple logging facade for java (SLF4J).
- You must not use the temporary disk space to write files or store data. This helps to avoid **no disk space error** at runtime.

Context

You use an adapter in an integration flow for outbound and inbound communication with different applications during runtime. In such scenario if an adapter does not support your communication requirements you use SAP ADK framework to create an adapter and deploy it in an integration flow. Here we describe how to build and deploy an adapter during runtime.

Procedure

1. Install the Adapter SDK for SAP Cloud Platform Integration. For more information, see Installation Instructions.
2. Open Eclipse.
3. Create a new adapter project **(with maven)** using the following substeps:
   a. Select *File* > *New* > *Project*.
   b. Choose *SAP Cloud Platform Integration* > *Adapter Project* in the *New Project* wizard.
   c. Choose *Next*.
   d. Enter details in the *New Adapter Project* wizard.

   > **i Note**
   >
   > If you are entering value in *Version* field, then you must enter the major, minor and macro versions. Currently, the tool supports only 1.0.0 version, which is also the default value.

   e. The *Enable Maven* is selected by default that identifies this project is of type Maven. You can change the component details as per your requirement.

   > **i Note**
   >
   > Deselect the *Enable Maven* option if you do not want to create a non-maven based custom adapter.

   f. Choose *Finish* to create a custom project.
   g. The following table describes different components created for the custom adapter project using maven:

   > **i Note**
   >
   > The advantage here is that you can develop UI and runtime components.

Component	Description
`*Component.java`	Sample runtime component.
`*Consumer.java`	Sample sender component.
`*Endpoint.java`	Logger information is found.

Component	Description
`*Producer.java`	Sample receiver component.
`metadata.xml`	UI values are coded here.
`*ComponentTest.java`	Sample JUnit test
`pom.xml`	Contains configuration details and dependencies.
`sap-sample`	Represents the endpoint scheme. **i Note** If you deploy multiple adapters make sure the file-names are unique. When you change the filename do the following: ○ In the endpoint class, change scheme value. For example, `@UriEndpoint(scheme = "sap-sample", syntax = "", title = "")`. ○ In `metadata.xml` change the attribute value. For example, `gen:RuntimeComponentBaseUri="sap-sample"`.

> **i Note**
>
> You can update to the latest ADK version by manually changing the version in `com.sap.cloud.adk.build.archive` artifact found in the `pom.xml` file. Refer the central maven repository to get the latest ADK version.

h. To run maven clean install right click-on the project, choose *Run As* > *Maven Build*.
i. In the *Goals* field, type **`clean install`** and *Run* the project.

> **i Note**
>
> To deploy the adapter project right click-on the adapter project, and choose *Deploy Adapter Project*.

4. Create a new adapter project **(without maven)** using the following substeps:
   a. Select *File* > *New* > *Project*.
   b. Choose *SAP Cloud Platform Integration* > *Adapter Project* in the *New Project* wizard.
   c. Choose *Next*.
   d. Enter details in the *New Adapter Project* wizard.

> **Note**
>
> If you are entering value in *Version* field, then you must enter the major, minor and macro versions. Currently, the tool supports only 1.0.0 version, which is also the default value.

e. Deselect the *Enable Maven* option if you do not want to build a custom adapter with maven.
f. Choose *Finish* to create the custom project.
g. Develop camel component.

> **Note**
>
> *createProducer()* or *createConsumer()* method, will throw *UnsupportedOperationException*, *RuntimeCamelException*, *OperationNotSupportedException* exceptions. You can prevent the generation of producer or consumer components by throwing *UnsupportedOperationException* exception.

h. Add one (mandatory) camel component `.jar` file in `component` folder and other additional (optional) dependent `.jar` files in `libs` folder.

> **Note**
>
> You must add only those jar files that are valid OSGi bundles. Component jar is a camel based component bundle. Dependent jars are the osgi bundles that are required by camel component.

i. Adapt the component metadata file according to the requirements of the new adapter type.

5. Right click-on the new adapter project or the component folder and select *Generate Metadata* > *Component Metadata* to generate a sample component metadata.

> **Note**
>
> Generate component metadata works only if annotations are marked in the camel bundle on endpoint class variables.
>
> **Annotations**:
>
> @Required : used to make the field mandatory and belongs to the package org.apache.camel.spi.Required
>
> **For example**: @UriParam
>
> @Required
>
> private String testField;
>
> Attribute behavior is set as *SecureAlias* if the variable name has **`password`**, **`key`**, **`secret`** or **`token`** keywords.

Follow the substeps to introduce custom classes using blueprint metadata that allows you to extend the runtime capabilities of the adapter:

1. Right click-on the new adapter project or the component folder and select *Generate Metadata* > *Blueprint Metadata* to generate a sample blueprint metadata.
2. Update the **`<ReferencedComponents>`** tag in `metadata.xml` to provide a reference to `bpMetadata.xml`.

> **Note**
>
> The value provided here is variant specific, for multiple variants fetch appropriate URL and update the `metadata.xml` in the matching variant. For example, a variant can be either Receiver or Sender.

6. You can define the following additional property values in `metadata.xml` file, for retrieving specific information on an adapter during runtime.

Value	Data Type	Endpoint Variable and Setter	Refers to...
`adapterInstanceID`	String	**`adapterInstanceID(string)`**	Participant ID, which helps to identify log error details.
`adapterVersion`	String	**`adapterVersion(string)`**	The adapter version.
`projectID`	String	**`projectID(string)`**	The integration flow project ID.
`userDefinedNamespaces`	Java.util.Map	**`userDefinedNamespaces(map<string,string>)`**	A namespace mapping generated at runtime for an integration flow model. It contains a namespace-prefix pair with a format **`xmlns:<prefix>=<namespace>`**.

> **Note**
>
> Use the **`<AdditionalMetadata>`** attribute, to include additional properties in the URI. You must place the attribute as an immediate child of any variant in the metadata file.
>
> **Sample Code**
>
> ```
> <Metadata>
>              <AdditionalMetadata>
>                    <Name>requiredIntegrationFlowProperties</Name>
>                    <Value>adapterInstanceID,  adapterVersion  ,
> projectID, userDefinedNamespaces </Value>
>                    <GuiLabels>
>                        <Label language="en"/>
>                    </GuiLabels>
>              </AdditionalMetadata>
>     </Metadata>
> ```

7. If you want to verify if the adapter project has valid contents such as component metadata in metadata folder and jar files in the libs and component folder, right click on the project and choose *Execute Checks*.
8. If you want to build your project locally, right click-on the project and choose *Deploy Adapter Project*.

> **Note**
>
> Once you build locally, the system creates *target.build* directory, where you can find a new adapter runtime archive.

9. To configure the Operations Server to connect your local Eclipse tooling to the tenant, execute the following substeps:
   a. Select *Window* > *Preferences* > *SAP Cloud Platform Integration* > *Operations Server*.
   b. Enter the server URL.
   c. Enter your user and password.
10. Deploy the adapter project.

> **Note**
>
> IFL worker node does not support ADK.

11. Check the status of your deployed project by executing the following substeps:
    a. In *Node Explorer*, choose worker node.

> **Note**
>
> You can construct an application URL in the following format **`https://<IFLMAP URL>/<web context path>/`** to call the servlet.

    b. Select the *Component Status View* tab and check the status of the adapter project.

> **Note**
>
> - If you do not delete an adapter but some other user deletes the adapter and still the metadata is available in integration flow, then you must restart the system.
> - You must undeploy the integration flows using the relevant adapter before undeploying adapter.

> **Note**
>
> You can implement logging in ADK framework by using Simple Logging Facade for Java (SLF4J) API package. Use the following maven coordinates in component bundle:
>
> **Sample Code**
>
> ```
> <dependency>
>       <groupId>org.slf4j</groupId>
>       <artifactId>slf4j-api</artifactId>
>       <version>[1.0,)</version>
> </dependency>
> ```
>
> Based on the log level configuration the log is visible only in the tail log.

Related Information

Component Metadata [page 12]
Blueprint Metadata [page 29]

2.3 Component Metadata

Component metadata defines the attributes of the sender and receiver adapters that contain multiple tabs. Each tab can contain multiple attributes. You can group the attributes into multiple attribute groups within the tabs. You can use the same attribute across the sender and receiver adapters by using attribute reference.

Component Metadata

This set of metadata allows you to define components details.

Component Metadata

Property	Datatype	Cardinality	Description
ComponentId	xsd:string	Attribute	Specifies a unique Id for the component. The following are technical identifiers and should not contain any space or special character. Example: `ctype::Adapter/cname::sap:dropbox/version::1.0.0` ctype is type of component, which is adapter in this particular example cname is component name, which is sap:dropbox in this example. Here, sap is vendor and dropbox is name
ComponentDisplayName	xsd:string	Attribute	Specifies display name of the component.
Variant		Child Element	Different variants for a specific component type - for example: Sender and Receiver variants of an adapter

Property	Datatype	Cardinality	Description

gen:RuntimeComponentBaseUri	xsd:string	Attribute	This attribute specifies the name with which the component is identified. If not added, then the baseUri value is taken from TP (Transport Protocol).
			
			Example for specifiying attribute in the CMD:
			
			Sample Code
			
			`<ComponentMetadata ComponentId="ctype::Adapter/ cname::sap:sampleadapter/version::1.0.0" ComponentName="sap:sampleadapter" UIElementType="Adapter" IsDefaultGenerator="true" hci:RuntimeComponentBaseUri="dropbox">`
			
			Example for generated bean.xml containing camel:from
			
			Sample Code
			
			`<camel:from id="MessageFlow_1_1435812482630" uri="dropbox:foo?path=/input/ test.txt&amp; method=get&amp;sendEmptyMessageWhenIdle=false&amp; initialDelay=1000&amp;greedy=false&amp;startScheduler=false&amp;timeUnit=MILLISECONDS&amp; useFixedDelay=false&amp;delay=500&amp;runLoggingLevel=TRACE"/>`

IsRequestResponse	xsd:string	Attribute	This attribute is used for enabling the component to allow the service call for Request Reply. Default value is false. Henceforth, you should define it at **variants** level only.
hci:FirstUriPart	xsd:string	Attribute	Specifies the initial part of the URI. Customer can provide custom values for the first part of the URI string. Default value is 'foo'.

Note

IsDefaultGenerator and Metadata Version must always be true and should never be set to false. Metadata Version attribute is generated with default value as 2.0.

Variants

This set of metadata allows you to define variants for a component.

Variants

<table>
<tr><th>Property</th><th>Data-type</th><th>Cardinal-ity</th><th>Description</th></tr>
<tr><td>Id</td><td>xsd:string</td><td></td><td>Provides a unique Id for the component variant.
Example: <code>ctype::AdapterVariant/cname::sap:dropbox/tp::dropbox/mp::dropbox/direction::Sender</code>
tp stands for transport protocol
mp stands for message protocol
direction refers to Sender or Receiver system</td></tr>
<tr><td>Groups</td><td></td><td>Child Ele-ment</td><td>Specifies collection of attributes defining metadata.</td></tr>
<tr><td>VariantDisplay-Name</td><td>xsd:string</td><td>Attribute</td><td>Specifies display name of the variant.</td></tr>
<tr><td>Input Content
Tag used <Input-Content></td><td></td><td>Child Ele-ment</td><td>Input content specifies the type of data the adapter processes.
<table>
<tr><th>Properties</th><th>Attributes/Child Ele-ments</th><th>Description</th></tr>
<tr><td>ContentType</td><td>Child Element</td><td>Actual content type
Supported content types: Xml, String, Boo-lean, Base64Encoded, Any, Json, Binary, Zip, GZip, Integer, NonXml, None and Text.</td></tr>
</table></td></tr>
<tr><td>Output Content</td><td></td><td>Child Ele-ment</td><td>Output content specifies the type of data the adapter produces. Example: XML.
<table>
<tr><th>Properties</th><th>Attributes/Child Ele-ments</th><th>Description</th></tr>
<tr><td>ContentType</td><td>Child Element</td><td>Actual content type
Supported content types: Xml, String, Boo-lean, Base64Encoded, Any, Json, Binary, Zip, GZip, Integer, NonXml, None and Text.</td></tr>
</table></td></tr>
</table>

Property	Data-type	Cardinality	Description

Metadata		Child Element	Helps to add additonal metadata in the component. Specifies the display name of transport protocol and message protocol.
			
			Example for specifiying attribute in the CMD:
			
			Sample Code
			
			`<Metadata>`
			`<AdditionalMetadata>`
			`<Name>mp</Name>`
			`<GuiLabels guid="e7fa4f1d-ec6a-4248-9752-920ea9616bc9" >`
			`<Label language="EN" >AS2</Label>`
			`<Label language="DE" >AS2</Label>`
			`</GuiLabels>`
			`</AdditionalMetadata>`
			`<AdditionalMetadata>`
			`<Name>tp</Name>`
			`<GuiLabels guid="e7fa4f1d-ec6a-4248-9752-920ea9616bc9" >`
			`<Label language="EN" >HTTPS</Label>`
			`<Label language="DE" >HTTPS</Label>`
			`</GuiLabels>`
			`</AdditionalMetadata>`
			`</Metadata>`
			
			Note
			
			You should refer message protocol as *mp* and transport protocol as *tp*.

ManifestEntries		Child Element	Specifies which manifest entries you can add in manifest file. You can add or update *Import-Package* entries in manifest files.
			
			Example for specifiying attribute in the CMD:
			
			Sample Code
			
			`<ManifestEntries>`
			`<ManifestEntry>`
			`<Name>Import-Package</Name>`
			
			`<Value>com.sap.gateway.core.service.ljs.registry;resolution:=optional,com.sap.gateway.core.ljs;resolution:=optional</Value>`
			`</ManifestEntry>`
			`</ManifestEntries>`

Property	Data-type	Cardinal-ity	Description

Allowed Headers		Child Ele-ment	Specifies which headers can be passed in the integration flow pipeline from sender component.
			Example for specifiying attribute in the CMD:
			Sample Code
			`<AllowedHeaders>`
			`Header1,Header2`
			`</AllowedHeaders>`

hci:FirstUriPart	xsd:strin g	Attribute	Specifies the initial part of the URI. Customer can provide custom values for the first part of the URI string.
IsRequestRes-ponse	xsd:strin g	Attribute	This attribute is used for enabling the variant to allow the service call for Re-quest Reply
supportsPolling	xsd:boo-lean	Attribute	This attribute depicts if the component has the polling behavior. (supportsPol-ling). It also helps in populating the timer icon at design time when IFlow is dec-orated. Default value is false. It is applicable for sender channel only.
AttachmentBe-havior	xsd:strin g	Attribute	This attribute helps to identify the component behavior towards attachment. • Modify - Create or update an attachment • Preserve - Pass through for the attachments • Drop - Attachment will be lost If attachment behavior is not specified; system considers that the component lost the attachment.

> **Note**
>
> If you want to rename a variant id or delete a variant from adapter, first undeploy the adapter with old variants and then deploy updated adapter.
>
> You can still view the older variants if you do not undeploy the adapter with old variants.

Tabs

This set of metadata allows you to specify tab pages for the adapter configuration user interface.

Tabs

Property	Datatype	Cardinality	Description
Id	xsd:string		Specifies the tab identifier.

Property	Datatype	Cardinality	Description

GuiLabels	xsd:string		Specifies a language-specific GUILabel. Sample GUILabel is shown below:
			`<GuiLabels>`
			`<Label language="EN">Address</Label>`
			`</GuiLabels>`
			Here, Label language = "EN" represents that the text of guilabel is in english. Address is the text for the attribute name.
			Currently, guilabel supports only English language.

Attribute Group

This set of metadata allows you to group adapter attributes.

Attribute Group

Property	Datatype	Cardinality	Description
Name	xsd:string		

Id	xsd:string		Specifies the attribute group identifier.
			Examples:
			`Dropbox Endpoint`
			`Dropbox Security Settings`

AttributeReference		Child Element	Specifies ReferenceName, ErrorMessage, Restriction, and Description
GuiLabels			Specifies a language-specific GUI label.

Property	Datatype	Cardinality	Description
Value - Datatype		Child Element	Datatype for value xsd:string xsd:integer xsd:boolean xsd:long xsd:password xsd:id xsd:idref **i Note** If an attribute is marked with *xsd:id* in metadata, then its value should be unique across elements of integration flow, during design time. If an attribute is marked with *xsd:idref* then it specifies a reference to another resource. If resource path starts with */<folder_name>/<resource_name>*, then checks will validate if specific resource path exists in *src\main\resource* folder of project or not.

The sample attribute group is shown below:

```
<AttributeGroup id = "Connection Details">
            <GuiLabels>
             <Label language="EN">Connection Details</Label>
            </GuiLabels>

           <AttributeReference>
             <ReferenceName>address</ReferenceName>
              <!-- Restriction tag defines java regular expression for field
value -->
             <Restriction>Constraint.isValidRegex([A-Za-z]+[0-9_~\\-\\.]*)</
Restriction>
             <!-- This tag defines message to be displayed if field value is
invalid -->
            <ErrorMessage>Enter a valid address. E.g. /file</ErrorMessage>
            <!-- Description tag defines tool tip for field value -->
            <description>Relative endpoint address on which..</description>
            </AttributeReference>
            <AttributeReference>
            <ReferenceName>wsdlURL</ReferenceName>
            <Restriction>Constraint.isValidRegex(^(https://){1}(.)+)</
Restriction>
            <ErrorMessage>Enter a valid URL. E.g. /wsdl/abc.wsdl;</ErrorMessage>
            <description>URL to the WSDL.</description>
            </AttributeReference>
            <AttributeReference>
             <ReferenceName>cxfOnewayRobust</ReferenceName>
             <description>Used for reliable one-way message exchanges.</
description>
            </AttributeReference>
</AttributeGroup>
```

Attribute Reference

Attribute Reference Table

Property	Data Type	Cardinality	Description
ReferenceName	xsd:string		Specifies that in the CMD there is an attribute definition associated with this reference name.
description	xsd:string		Brief description of the attribute, which is shown as the tool tip.
Restriction	xsd:string		Details given below in the table.
ErrorMessage	xsd:string		If the constraint specified in the restriction tag fails, the system shows the error message set in this tag. If this field is not set, then the system shows a default error message at the time of check failure.

List of Constraints that are supported for Restriction tag

Constraint	Description

isValidRegex	Method will check a value against a passed regex. Returns true in case value matches regex
	Sample Code
	`<AttributeReference>`
	`<ReferenceName>stepid</ReferenceName>`
	`<Restriction>Constraint.isValidRegex([a-zA-Z0-9]+)</Restriction>`
	`<ErrorMessage>StepID not defined for Persist Message step</ErrorMessage>`
	`</AttributeReference>`

isAlphaNumeric	Checks whether the pass value is alphanumeric
isStartsWithLetter	Checks the passed value starts with Letter
isValidURIString	Returns true if it is valid URI String. It can start with http(s), ftp, file or ldap

Constraint	Description
isValidXMLString	Returns true if passed string is a valid XML String
isValidNCName	Return true if passed string value is valid NCName
isValidXpath	Returns true if passed string value is valid xpath expression

Attribute Metadata

This set of metadata allows you to define the details of all adapter attributes. This includes the definition of fixed values for input help, and the kind of user interface element used for an adapter attribute (for example if it is a checkbox or a dropdown list).

The sample attribute metadata should be as shown below:

```
<AttributeMetadata>
              <Name>address</Name>
              <Usage>true</Usage>
              <Default>/BSN/paymentOrder</Default>
              <DataType>xsd:string</DataType>
              <isparameterized>true</isparameterized>
              <Length>200</Length>
              <FixedValues>
                     <FixedValue>
                           <Value>Atleast Once</Value>
                           <GuiLabels>
                                  <Label language="EN">Atleast Once</Label>
                           </GuiLabels>
                     </FixedValue>
              </FixedValues>
              <GuiLabels>
                     <Label language="EN">Address</Label>
              </GuiLabels>
       </AttributeMetadata>
```

Attribute Metadata

Property	Datatype	Cardinality	Description
Name	xsd:string		Specifies the parameter id. Example: `address`
Usage	xsd:boo-lean		Specifies if the field is mandatory or not. If the value is true then usage of field is mandatory. If the value is false then usage of field is not mandatory.
Default	xsd:string		Specifies a default value. Example: `/BSN/paymentOrder`

Property	Datatype	Cardinality	Description

Datatype			Specifies the datatype for the value.
			Examples:
			`xsd:string`
			`xsd:integer`
			`xsd:choice`
			If the value has data type <xsd:choice> then it is treated as Radio Button.

Length	xsd:integer		Specifies the maximum character length for field values.
Minlength	xsd:integer	Child Element	Specifies the minimum character length for field values.
HelpService		Child Element	Helps to configure specified resources in adapters. Example for specifiying attribute in the CMD: **Sample Code** `<HelpService> <HelpServiceName> ResourceLookupService </HelpServiceName> <HelpServiceQueryAttributes> <HelpServiceQueryAttribute> <Name> ResourceType </Name> <Value> xml </Value> </HelpServiceQueryAttribute> <HelpServiceQueryAttribute> <Name> ResourceLocation </Name> <Value> edmx </Value> </HelpServiceQueryAttribute> </HelpServiceQueryAttributes> </HelpService>` Once you add the above code, then a *Browse* button gets enabled. You can click on the button and select resource file from the project of file system. *ResourceType* specifies resource extension. Currently, only one extension can be specified. *ResourceLocation* specifies relative path to project from where files are read or written.
GUILabel		Child Element	Specifies a language-specific GUI label.

<table>
<tr><th>Property</th><th>Datatype</th><th>Cardinality</th><th>Description</th></tr>
<tr><td>Fixed-Value(s)</td><td></td><td>Child Element</td><td>Specifies a fixed list of allowed values.

<table><tr><th>Properties</th><th>Attributes/Child Elements</th><th>Description</th></tr><tr><td>FixedValue</td><td>Child Element</td><td>Specifies fixed list of allowed values. Multiple FixedValue tags together are called a combo.</td></tr><tr><td>isEditable</td><td>Child Element</td><td>If the element is set to 'true', then the dropdown and combo fields are editable.</td></tr><tr><td>Sort</td><td>Child Element</td><td>If the element is set to 'true', then the values are arranged either in ascending or descending order. Use the following formats to arrange the values:
• To arrange the values in ascending order, use <sort order=ascending>true</sort>
• To arrange the values in descending order, use <sort order="descending">true</sort>

i Note
If the element is set to 'false', then the values are arranged as per the order defined in the cmd.</td></tr></table></td></tr>
</table>

Attribute Reference Table

Property	Data Type	Cardinality	Description
AttributeName		Child Element	Name of the attribute
AttributeValue		Child Element	Value of the attribute

Property	Data Type	Cardinality	Description
AndCondition		Child Element	Attribute Reference Property This is complex constraint that is used when there are two or more complex or basic constraints that need to be satisfied to achieve certain criteria. Example of Complex condition Sample Code `<AndCondition><EditCondition><AttributeName>reference1</AttributeName><AttributeValue>reference1value</AttributeValue></EditCondition><EditCondition><AttributeName>reference2</AttributeName><AttributeValue>reference2value</AttributeValue></EditCondition></AndCOndition>` As you can see in the above example AndCondition is complex condition that contains two or more EditConditions(basic conditions) and for the condition to be satisfied both of these basic conditions is satisfied.

Property	Data Type	Cardinality	Description
OrCondition		Child Element	Attribute Reference Property This is complex constraint that is used when satisfaction of any one of the nested complex or basic constraint is required to achieve certain criteria The structure of OrCondition is similar to And condition, with only change from <AndCondition> to <OrCondition>.

Property	Data Type	Cardinality	Description
NotCondition		Child Element	Attribute Reference Property This is complex constraint that is mostly used with other complex constraints, this constraint reverses the result returned by the nested complex or basic condition Not condition can contain only one edit condition but multiple complex conditions. These complex conditions can be used with each other. Sample Code <AndCondition> <OrCondition> <EditCondition ></ EditCondition> <EditCondition ></ EditCondition> </ OrCondition> <NotCondition> <EditCondition ></ EditCondition> </ NotCondition> </ AndCondition>

Property	Data Type	Cardinality	Description
EditCondition		Child Element	Attribute Reference Property Constraint to be executed, this is on top of base constraint defined at Attribute Level Environment Variables: If the Attribute Name starts with $ then next part is considered as an Environment variable. There can be two types of variables: a) $PROFILE.NAME - environment variables starting with profile are properties fetched on profile configured for project b) $ENV.VARIABLE - environment variables starting with env are system properties for vm
AttributeBehavior	xsd:string	Child Element	If you assign *AttributeBehavior* as **`SecureAlias`** for a particular field, then the field refers to alias of security parameter artifact.

Attribute Table Metadata

Property	Data Type	Cardinality	Description
Name	xsd:string		
Usage	xsd:boolean		

Property	Data Type	Cardinality	Description
HelpService		Child Element	Helps to configure ceritificate in adapters. Example for specifiying attribute in the CMD: **Sample Code** `<HelpService> <HelpServiceName> CertificateLookupService </HelpServiceName> </HelpService>` Once you add the above code, then a *Browse* button gets enabled. You can click on the button and select certificate file. This attribute supports **.crt* and **.cer* types of certificates only.
AttributeReference			Specifies reference name that is header of the column. **Note** Each reference name constitutes to one column in the table.
Guilabels	xsd:string		Specifies the label of the table.

The sample attribute table metadata should be as shown below:

Sample Code

```
<AttributeTableMetadata>
            <Name>clientCertificates</Name>
            <Usage>true</Usage>
            <GuiLabels guid="01c2bd59-8dd1-4efb-b795-869851eb979b">
                <Label language="EN">Client Certificate Authorization</Label>
                <Label language="DE">Client Certificate Authorization</Label>
            </GuiLabels>
```

```
            <AttributeReference>
                <ReferenceName>clientCertificate.subjectDN</ReferenceName>
            </AttributeReference>
            <AttributeReference>
                <ReferenceName>clientCertificate.issuerDN</ReferenceName>
            </AttributeReference>
        </AttributeTableMetadata>
```

i Note

- You must not use any special character in the table fields.
- You can reload component metadata in the following 3 ways:
  - You can reopen eclipse tool.
  - You can reconnect to server.
  - You can use  icon (on eclipse toolbar) to manually download it.

2.4 Blueprint Metadata

Blueprint metadata defines the attributes of the sender and receiver adapters that contain multiple tags.This document mainly describes all the tags and attributes apart from CMD and camel blueprint tags for providing additional generation injection for ADK development.

Blueprint Metadata

This set of tags and attributes allows you to define the metadata.

Attributes	Datatype	Cardinality	Description
createOnce	xsd:string		If you set the string value as “true”, then the bean instantiation in the blueprint generation is done only once. By default the value is 'false'.

Attributes	Datatype	Cardinality	Description
createCondition	xsd:string		This attribute can be used to conditionally instantiate the bean. The syntax followed is similar to XSLT test attribute. For example: **Sample Code** `<bean id="123" class="com.example.Someclass.java" createCondition="$.cmd.attribute(proxyType)='onPremise'" />` Here `Someclass.java` is generated in the blueprint only if the UI variable `proxyType` contains the value `onPremise`.

Tag	Description
`<TransformationTags xmlns:bp="http://www.adk.gen/xmlns/blueprint/">`	This is the parent tag for the `<blueprint>` tag. This tag does not contain any generation specific information as part of attribute. The `<blueprint>` tag must be included within this tag.
`<bp:blueprint>`	This tag can be used to specify additional namespaces if needed.
`<bp:bean>`	Beans are declared using <bean> element.
`<bp:reference>`	To generate References this tag can be used. This tag does not contain any new attributes. It is similar to the camel blueprint's reference tag. Refer the XSD above for all supported attributes.
`<cxfcore:bus xmlns:cxfcore="http://cxf.apache.org/blueprint/core">`	Refer `http://cxf.apache.org/schemas/blueprint/core.xsd` for all attributes supported.
`<cxf:cxfEndpoint xmlns:cxf="http://camel.apache.org/schema/blueprint/cxf">`	Refer `https://camel.apache.org/schema/cxf/camel-cxf.xsd` for all the attributes supported.

Tag	Description
`<camel:camelContext xmlns:camel=”http://camel.apache.org/schema/blueprint”>`	This is the root camel context tag just for the reference. This contains all the properties a final blueprint's camelContext contains as part of g&b generation. The user is not allowed to make any change to this tag. Any changes/additions will simply be ignored.
`<http-conf:conduit xmlns:http-conf="http://cxf.apache.org/transports/http/configuration" >`	This is same as that of Camel Blueprint specification for conduit tag. Refer `http://cxf.apache.org/schemas/configuration/http-conf.xsd`.
`<camel:route xmlns:camel="http://camel.apache.org/schema/blueprint">`	This is same as that of Camel blueprint specification <route>. Must apear within <camelContext>.. Refer the XSD for the attributes. Multiple routes are supported.
`<properties>`	Can be contained in the DSL just for reference. This tag must be same as g&b's generated tag. It has no significance. i.e. any modification to this tag can simply be ignored by the DSL processor.
`<streamCaching>`	Provided just for the reference.

2.5 Enabling Endpoint Visualization for Custom Adapter

Configure a custom adapter with endpoint visualization to view the adapter endpoint URL.

The following sections describe the guidelines that you need to understand and follow:

Adding Libraries

You can find the latest ADK API in the Adapter Development Kit section. After downloading the `*.zip` files, add the following libraries to the *src* > *main* > *resources* folder that is in your ADK project:

- `com.sap.it.public.adapter.api.jar`
- `com.sap.it.public.generic.api.jar`

For successful code compilation, right-click the `*.jar` file and choose *Build Path* > *Add to Build Path* to add the jar files to the `Referenced Libraries` of the adapter project.

Adding Dependencies

Open the `pom.xml` file and add the dependencies shown here:

Source Code

```
<dependency>
<groupId>com.sap.it.public</groupId>
<artifactId>adapter.api</artifactId>
<version>...</version>
<scope>system</scope>
<systemPath>${project.basedir}/src/main/resources/adapter.api.<version of the
downloaded file>.jar</systemPath>
</dependency>
<dependency>
<groupId>com.sap.it.public</groupId>
<artifactId>generic.api</artifactId>
<version>...</version>
<scope>system</scope>
<systemPath>${project.basedir}/src/main/resources/generic.api.<version of the
downloaded file>.jar</systemPath>
</dependency>
```

Note

Make sure that you use the latest version of the `*.jar` files.

After adding the dependency, add the `com.sap.it.public` group ID to the `<excludeGroupIds>` tag.

Adding Endpoint Information Service

You must create a **Class** and use the following method to retrieve the list of all endpoints for all integration flows:

Source Code

```
import java.util.List;
import com.sap.it.api.adapter.monitoring.AdapterEndpointInformation;
import com.sap.it.api.adapter.monitoring.AdapterEndpointInformationService;
public class AdapterEndpointInfo implements AdapterEndpointInformationService{
    /**
     * Gets the list of all endpoints for all IntegrationFlows used by the
     * adapter
     *
     * @return the list of endpoints
     */
    @Override
    public List<AdapterEndpointInformation> getAdapterEndpointInformation() {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public List<AdapterEndpointInformation>
getAdapterEndpointInformationByIFlow(String arg0) {
        // TODO Auto-generated method stub
        return null;
    }
```

```
}
```

The service implementation must contain the following method, which supports the retrieval of all endpoints exposed across all integration flows:

Source Code

```
package com.sap.it.api.adapter.monitoring;
import java.util.List;
/**
 * Interface, which adapter developers have to implement to return the
endpoints
 * (URLs) used by their adapter
 */
public interface AdapterEndpointInformationService {
    /**
     * Gets the list of all endpoints for all IntegrationFlows used by the
     * adapter
     *
     * @return the list of endpoints
     */
    public List<AdapterEndpointInformation>
getAdapterEndpointInformationList();
    /**
     * Gets the list of endpoints for a given IntegrationFlow. If the given
     * Integration Flows does not use the adapter, it has to return an empty
     * instance list.
     *
     * @param integrationFlowId
     *            the id (bundle symbolic name) of the Integration Flow
     * @return the endpoint information for the given Integration Flow
     */
    public AdapterEndpointInformation getAdapterEndpointInformation(String
integrationFlowId);
}
```

The class describing the **adapter endpoint information** is mentioned here:

Source Code

```
   package com.sap.it.api.adapter.monitoring;
import java.util.List;
/**
 * Class representing endpoint information for an Integration Flow
 *
 */
public class AdapterEndpointInformation {
    [..]

    /**
     * Get the endpoint instance list for this Integration Flow.
     *
     * @return the endpoint instance list
     */
    public List<AdapterEndpointInstance> getAdapterEndpointInstanceList()
    [..]
    /**
     * Set the endpoint instance list for this object.
     *
     * @param adapterEndpointInstanceList
     *            the endpoint instance list
     */
```

```
    public void setAdapterEndpointInstanceList(List<AdapterEndpointInstance>
adapterEndpointInstanceList)
    [..]
    /**
     *
     * @return the integration flow id (bundle symbolic name)
     */
    public String getIntegrationFlowId()
    [..]
    /**
     * sets the integration flow id
     *
     * @param integrationFlowId
     *            the integration flow id (bundle symbolic name)
     */
    public void setIntegrationFlowId(String integrationFlowId)
    [..]
}
```

The class describing the **adapter endpoint instance** is mentioned here:

Source Code

```
   package com.sap.it.api.adapter.monitoring;
/**
 * The endpoint instance consists of an <code>endpointCategory</code> which
 * describes the kind of endpoint and an URL relative to the worker node. It
is
 * mainly about the distinction between endpoints to which messages can be
sent
 * and API definition endpoints from which metadata about the API can be
 * retrieved.<br/>
 * The consumer of the API is responsible for retrieving the correct
dispatcher
 * URL for the worker node (group) on which the Integration Flow is
 * deployed.<br/>
 * There is also an optional attribute available to provide additional info
 * about the character of the endpoint in case the category is not sufficient.
 * For CXF based endpoints, the WSDL download is offered in several flavors,
for
 * instance, so the adapter can provide this information in that attribute.
 */
public class AdapterEndpointInstance {
    [..]
    /**
     * instantiates the object
     *
     * @param endpointCategory
     *            the endpoints category, see also {@link EndpointCategory}.
     * @param relativeEndpointUrl
     *            the URL relative to the worker nodes dispatcher URI starting
     *            with a forward slash ('/')
     */
    public AdapterEndpointInstance(EndpointCategory endpointCategory, String
relativeEndpointUrl)
    [..]
    /**
     * instantiates the object
     *
     * @param endpointCategory
     *            the endpoints category, see also {@link EndpointCategory}.
     * @param relativeEndpointUrl
     *            the URL relative to the worker nodes dispatcher URI starting
     *            with a forward slash ('/')
     * @param additionalInfo
     *            additional information (optional) in order to provide more
```

```
     *            information if the category is not self explanatory. For
example, in
     *            case the WSDLs are offered in different flavors for
different
     *            consumers
     */
    public AdapterEndpointInstance(EndpointCategory endpointCategory, String
relativeEndpointUrl, String additionalInfo)
    [..]
    /**
     *
     * @return the endpoints category
     */
    public EndpointCategory getEndpointCategory()
    [..]

    /**
     *
     * @return the URL relative to the worker nodes dispatcher URI starting
with
     *         a forward slash ('/')
     */
    public String getRelativeEndpointUrl()
    [..]

    /**
     *
     * @return additional information (optional) in order to provide more
     *         information if the category is not self explanatory. For
example, in case
     *         the WSDLs are offered in different flavors for different
     *         consumers. Can be null
     */
    public String getAdditionalInfo()
    [..]
}
```

The `EndpointCategory` can have the following values:

Source Code

```
package com.sap.it.api.adapter.monitoring;
public enum EndpointCategory {
    ENTRY_POINT, DEFINITION_OAS_JSON, DEFINITION_OAS_YAML,
DEFINITION_RAML_JSON, DEFINITION_RAML_YAML, DEFINITION_WSDL, DEFINITION_EDMX
}
```

Binding the Service with the Implementation

Next, you link the implementation to the service by creating a `beans.xml` file in the `src/main/resources/OSGI-INF/blueprint/` path and adding the following snippet:

Source Code

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0 http://
www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">
```

```
    <bean id="iAdapterEndpointInformationService" class="<add the
implementation class path>"
        activation="eager"></bean>
    <service id="IAdapterEndpointInformationService"
ref="iAdapterEndpointInformationService"

interface="com.sap.it.api.adapter.monitoring.AdapterEndpointInformationService
">
    </service>
</blueprint>
```

To support the binding snippet, add the following dependency in the `pom.xml` file:

Source Code

```
<dependency>
<artifactId>camel-blueprint</artifactId>
<groupId>org.apache.camel</groupId>
<version>${camel.version}</version>
</dependency>
```

Updating the Dependency to Export the Service

Ensure that the following dependency version is always greater than or equal to **1.25.0.** to export this service from your subsystem.

Source Code

```
<dependency>
<groupId>com.sap.cloud.adk</groupId>
<artifactId>com.sap.cloud.adk.build.archive</artifactId>
<version>1.25.0</version>
</dependency>
```

Deploying the Adapter

After completing the process, deploy the adapter and use the Cloud Integration Web application to consume the adapter in an integration flow. Now, go to the *Manage Integration Content* section to verify the endpoint information.

2.6 Enabling Authentication for Servlet Based Adapters

Context

Both Basic and Certificate based authentication can be enabled by performing the steps described here:

Procedure

1. Open the `web.xml` file and add the filter definition given below:

   Source Code

   ```
   <filter>
       <filter-name>ADKAuthenticationFilter</filter-name>
       <filter-
   class>com.sap.it.api.asdk.cloud.authentication.ADKAuthenticationFilter</
   filter-class>
   </filter>
   <filter-mapping>
       <filter-name>ADKAuthenticationFilter</filter-name>
       <url-pattern>/*</url-pattern>
   </filter-mapping>
   ```

2. Open the `pom.xml` file and include the below import package instruction in the maven-bundle-plugin:

   Source Code

   ```
   <Import-Package>com.sap.it.api.asdk.cloud.authentication,*</Import-Package>
   ```

3. Save the changes.

Results

Now you can connect to the adapter's endpoint either using Basic or Certificate based authentication.

2.7 Enabling Tracing for Custom Adapter

Configure a custom adapter with a tracing feature that records the processing of the incoming and outgoing messages in an adapter.

In the integration flow model, the logs generated by the adapter tracing help you to do the following:

- Monitor the overall status of the incoming or outgoing messages
- Troubleshoot specific errors encountered by the adapter during runtime

> **i Note**
>
> The incoming or outgoing messages in an adapter are only recorded if you have set the log level to *Trace* in the Cloud Integration Web application.

Adding Libraries

You can find the latest ADK API in the Adapter Development Kit section. After downloading the `*.zip` files, add the following libraries to the *src* > *main* > *resources* folder that is in your ADK project:

- `com.sap.it.public.adapter.api.jar`
- `com.sap.it.public.generic.api.jar`

For successful code compilation, right-click the `*.jar` file and choose *Build Path* > *Add to Build Path* to add the jar file to the `Referenced Libraries` of the adapter project.

Adding Dependencies

Open the `pom.xml` file and add the following dependencies to enable the tracing feature:

> **Source Code**

```
<dependency>
    <groupId>com.sap.it.public</groupId>
    <artifactId>generic.api</artifactId>
    <version>2.12.0</version>
    <scope>system</scope>
    <systemPath>${project.basedir}/src/main/resources/
com.sap.it.public.generic.api-2.12.0.jar</systemPath>
</dependency>
<dependency>
    <groupId>com.sap.it.public</groupId>
    <artifactId>adapter.api</artifactId>
    <version>2.12.0</version>
    <scope>system</scope>
    <systemPath>${project.basedir}/src/main/resources/
com.sap.it.public.adapter.api-2.12.0.jar</systemPath>
</dependency>
```

> **i Note**
>
> Make sure that the version of the `*.jar` files is greater than **2.12.0**.

After adding the dependency, add the `com.sap.it.public` group ID to the `<excludeGroupIds>` tag.

Enabling Tracing

The tracing feature is only relevant for adapters that transform the message either before sending or upon reception.

> **→ Recommendation**
>
> - Inbound messages are traced before the transformation takes place, which means that the user can see the incoming message with its original payload and headers. Outbound messages (headers and payload) are traced after the transformation has taken place.
> - The trace data is written even if errors occur. This can help the user to analyze transformation errors for inbound messages or connection errors for outbound messages.
> - The inbound message is traced after authentication and authorization have taken place. This is to prevent denial-of-service attacks that flood the trace database table.
> - Security-relevant header values must be obfuscated.
> - Trace data is only written if the adapter trace is enabled for the integration flow.
> - If possible, also set the character encoding for the trace payload.

The following procedure helps you to implement adapter tracing in either Consumer or Producer component:

1. Use the following code snippet to retrieve the `Adapter Message Log Factory` from your endpoint:

   Source Code

   ```
   /** get Adapter Message Log Factory */
    AdapterMessageLogFactory msgLogFactory = (AdapterMessageLogFactory)
   this.endpoint.getCamelContext().getRegistry()
   .lookupByName(com.sap.it.api.msglog.adapter.AdapterMessageLogFactory.class.
   getName());
   ```

2. Use the following code snippet to create the `Adapter Message Log` instance from the `Adapter Message Log Factory`:

   Source Code

   ```
   AdapterMessageLog mplLog =
   adapterMessageLogFactory.getMessageLog(exchange, description, "<CMD
   component ID of the adapter>",
               UUID.randomUUID().toString());
   ```

3. Use the snippet to create the trace message:

   Source Code

   ```
   AdapterTraceMessage traceMessage = mplLog.createTraceMessage(type, body,
   truncated );
   ```

4. Use the snippet to write the trace message to the MPL log:

Source Code

```
mplLog.writeTrace(traceMessage);
```

The interfaces available for `Adapter Message Log` are:

```
/**
    * Returns <code>true</code> if the adapter tracing for the integration
flow to
    * which this message log belongs is active, otherwise <code>false</code>.
    */
   boolean isTraceActive();

   /**
    * Creates a trace message which can be used in the method
    * {@link #writeTrace(AdapterTraceMessage)}.
    *
    * The supplied message body shall not exceed 25 MB = 25 * 1024 * 1024
bytes =
    * 26214400 bytes. We strongly recommend to supply a truncated copy of the
    * original message body if the original message body exceeds 25 MB. If,
despite
    * of this rule, the supplied message body still exceeds 25 MB, then the
body
    * will be truncated internally and the property <tt>isBodyTruncated</tt>
will
    * be set to <code>true</code>.
    *
    * @param traceMessageType
    *            type of the trace message
    * @param body
    *            message body, shall not exceed 25 MB, truncate the body if
    *            necessary
    * @param isBodyTruncated
    *            indicator whether the supplied message body is truncated
    * @throws IlleagalArgumentException
    *             if <tt>traceMessageType</tt> or <tt>body</tt> is
    *             <code>null</code>
    *
    */
   AdapterTraceMessage createTraceMessage(AdapterTraceMessageType
traceMessageType, byte[] body, boolean isBodyTruncated);

   /**
    * Writes a trace message into the adapter trace if the trace for the
    * integration flow of the adapter is active.
    * <p>
    * <b>Attention:</b>Camel endpoints, which are used in several from
elements of
    * different Camel routes cannot write traces. This method will not have
any
    * effect for such kind of endpoints.
    * <p>
    * Messages incoming to the adapter (inbound messages) shall be traced, if
the
    * adapter transforms the message before it is handled over to the Camel
route;
    * the trace shall be written <b>before</b> the transformation takes place.
    * Transformation here means changes to the message body or headers.
    * <p>
    * Messages outgoing from the adapter (outbound messages) shall be traced
if the
    * adapter transforms the message before it is sent out; the trace shall be
    * written <b>after</b> the transformation.
```

```
     * <p>
     * Typical usage:
     *
     * <pre>
     * {@code
     *    if (this.isTraceActive()){
     *          byte[] body = getTraceBody(originalMessage);
     *          boolean isBodyTruncated =
isBodyTruncated(body,originalMessage);
     *          // specify the correct adapter message type, in this example
we use AdapterTraceMessageType.SENDER_INBOUND
     *          AdapterTraceMessage traceMessage =
this.createTraceMessage(AdapterTraceMessageType.SENDER_INBOUND, body,
isBodyTruncated);
     *          // fill the optional properties of the trace message: encoding
and headers
     *          traceMessage.setEncoding(getEncoding());
     *          traceMessage.setHeaders(getHeaders());
     *          this.writeTrace(traceMessage);
     *    }
     * </pre>
     *
     * @param traceMessage trace message
     */
   void writeTrace(AdapterTraceMessage traceMessage);
```

Write the trace data. You can use the following example for reference:

Example

Sample Code

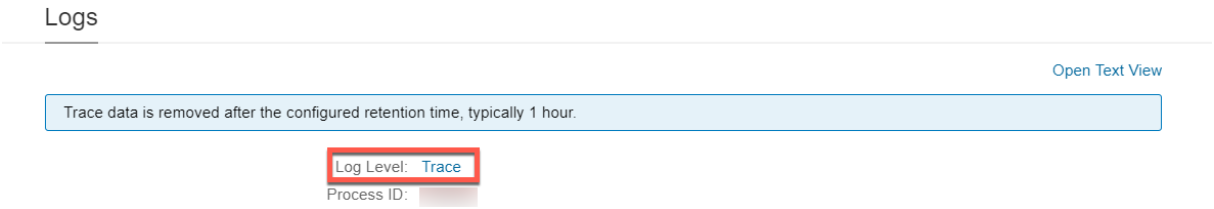
```
import org.apache.camel.Exchange;
import com.sap.it.api.msglog.adapter.AdapterMessageLog;
import com.sap.it.api.msglog.adapter.AdapterMessageLogFactory;
import com.sap.it.api.msglog.adapter.AdapterTraceMessage;
import com.sap.it.api.msglog.adapter.AdapterTraceMessageType;

// OSGi service
private final AdapterMessageLogFactory adapterMessageLogFactory;

void writeTrace(Exchange exchange, byte[] traceData, boolean
isOutbound) {
    // replace "<adapter type>" by your adapter type!
    String text = isOutbound ? "Sending <adapter type> message" :
"Receiving <adapter type> message";
    // replace "<CMD component ID of the adapter>" by your CMD
component ID
    AdapterMessageLog mplLog =
adapterMessageLogFactory.getMessageLog(exchange, text, "<CMD component
ID of the adapter>",
             UUID.randomUUID().toString());
    if (!mplLog.isTraceActive()) {
        return;
    }
    // if you have a fault inbound message then specify
AdapterTraceMessageType.RECEIVER_INBOUND_FAULT,
    // if you have a fault outbound message then specify
AdapterTraceMessageType.SENDER_OUTBOUND_FAULT
    // for synchronous adapters you may also need
AdapterTraceMessageType.SENDER_OUTBOUND and
AdapterTraceMessageType.RECEIVER_INBOUND
    AdapterTraceMessageType type = isOutbound ?
AdapterTraceMessageType.RECEIVER_OUTBOUND :
AdapterTraceMessageType.SENDER_INBOUND;
```

```
    AdapterTraceMessage traceMessage = mplLog.createTraceMessage(type,
traceData, false );//Setting isTruncated as false assuming traceData is
less than 25MB.
    // Encoding is optional, but should be set if available.
    traceMessage.setEncoding("UTF-8");
    // Headers are optional and do not forget to obfuscate security
relevant header values.
    mplLog.writeTrace(traceMessage);
}
```

5. Build and deploy the adapter.

Viewing the Trace Data

To show the trace information details, select an integration flow from the overview list, go to the *Log* section, and open the link by clicking the *Trace*. For more information, see .



2.8 Accessing User Credentials

Context

Follow the procedure below to get user credentials in an ADK project.

Procedure

1. Download the latest ADK API from the Adapter Development Kit section.
2. Add the `com.sap.it.public.generic.api.jar` file to the `resources` folder that is in your ADK project.

3. Open the `pom.xml` file and add the dependency below:

Source Code

```
<dependency>
 <groupId>com.sap.it.public</groupId>
 <artifactId>api</artifactId>
 <version>2.8.0</version>
 <scope>system</scope>
 <systemPath>${project.basedir}/src/main/resources/
com.sap.it.public.generic.api-2.8.0.jar</systemPath>
</dependency>
```

4. Add the `com.sap.it.public` group ID to the `<excludeGroupIds>`.
5. To import user credentials, use the code below:

Source Code

```
SecureStoreService secureStoreService =
ITApiFactory.getService(SecureStoreService.class, null);
UserCredential userCredential = secureStoreService.getUserCredential();
/**
*Provide an alias name in getUserCredential()
*/
```

Add the following import declarations:

Source Code

```
import com.sap.it.api.securestore.SecureStoreService;
import com.sap.it.api.securestore.UserCredential;
import com.sap.it.api.securestore.exception.SecureStoreException;
import com.sap.it.api.ITApiFactory;
```

2.9 Accessing Trust and Key Managers

Context

Follow the procedure below to get trust and key managers in an ADK project.

Procedure

1. Download the latest ADK API from the Adapter Development Kit section.

2. Add the `com.sap.it.public.generic.api.jar` file to the `resources` folder that is in your ADK project.
3. Open the `pom.xml` file and add the dependency below:

Source Code

```
<dependency>
 <groupId>com.sap.it.public</groupId>
 <artifactId>api</artifactId>
 <version>2.8.0</version>
 <scope>system</scope>
 <systemPath>${project.basedir}/src/main/resources/
com.sap.it.public.generic.api-2.8.0.jar</systemPath>
</dependency>
```

4. Add the `com.sap.it.public` group ID to the `<excludeGroupIds>`.
5. Depending on your requirements, add the code below to either the `*Producer.java` (receiver adapter) or `*Consumer.java` (sender adapter) file. This allows you to import all the trust managers.

Source Code

```
KeystoreService keyStoreService =
ITApiFactory.getService(KeystoreService.class, null);
TrustManager[] trustmanager = keyStoreService.getTrustManagers();
/**
*Provide an alias name in getTrustManagers()
*/
```

Add the following import declarations:

Source Code

```
import javax.net.ssl.TrustManager;
import com.sap.it.api.ITApiFactory;
import com.sap.it.api.exception.InvalidContextException;
import com.sap.it.api.keystore.KeystoreService;
import com.sap.it.api.keystore.exception.KeystoreException;
```

6. To import all the key managers, use the code below:

Source Code

```
KeystoreService keyStoreService =
ITApiFactory.getService(KeystoreService.class, null);
KeyManager[] keymanagers = keyStoreService.getKeyManagers();
/**
*Provide an alias name in getKeyManagers()
*/
```

Add the following declaration:

Source Code

```
import javax.net.ssl.KeyManager;
```

2.10 Accessing On-Premise Application using Cloud Connector

Follow the steps here to extend the connectivity of SAP Cloud Platform Integration with remote on-premise application.

Context

You can prepare an Adapter Development Kit (ADK) project to access any on-premise application using Cloud Connector.

Procedure

1. Download the latest ADK API from the Adapter Development Kit section.
2. Add the `com.sap.it.public.generic.api.jar` file to the `resources` folder that is in your ADK project.
3. Open the `pom.xml` file and add the dependencies below:

   Source Code

   ```
   <dependency>
    <groupId>com.sap.it.public</groupId>
    <artifactId>generic.api</artifactId>
    <version>2.X.X</version>
    <scope>system</scope>
    <systemPath>${project.basedir}/src/main/resources/
   com.sap.it.public.generic.api-2.X.X.jar</systemPath>
   </dependency>
   ```

   Source Code

   ```
   <dependency>
   <groupId>com.sap.it.public</groupId>
   <artifactId>adapter.api</artifactId>
   <version>2.X.X</version>
   <scope>system</scope>
   <systemPath>${project.basedir}/src/main/resources/
   com.sap.it.public.adapter.api-2.X.X.jar</systemPath>
   </dependency>
   ```

4. Add the `com.sap.it.public` group ID to the `<excludeGroupIds>`.
5. Depending on your requirements, add the code below to either the `*Producer.java` (receiver adapter) or `*Consumer.java` (sender adapter) file. This is for connecting to on-premise application.

Source Code

Use the below code to fetch the proxy details and additional headers.

```
CloudConnectorContext context = new CloudConnectorContext();
context.setConnectionType(ConnectionType.HTTP);
CloudConnectorProperties props =
ITApiFactory.getService(CloudConnectorProperties.class, context);
String proxyHost = props.getProxyHost();
String proxyPort = props.getProxyPort();
Map<String, String> additionalHeaders = props.getAdditionalHeaders();
```

Add the following import declarations:

Source Code

```
import com.sap.it.api.ITApiFactory;
import com.sap.it.api.ccs.adapter.CloudConnectorContext;
import com.sap.it.api.ccs.adapter.CloudConnectorProperties;
import com.sap.it.api.ccs.adapter.ConnectionType;
```

Sample Code

The example here shows how you can use the above proxy details in HTTP client. Ensure to provide the location ID of the Cloud Connector using the header **`SAP-Connectivity-SCC-Location_ID`**.

```
DefaultHttpClient httpclient = new DefaultHttpClient();
HttpGet httpGet = new HttpGet(address);
HttpHost proxy = new HttpHost(proxyHost, proxyPort);
httpclient.getParams().setParameter(ConnRoutePNames.DEFAULT_PROXY, proxy);
httpGet.setHeader("SAP-Connectivity-SCC-Location_ID", locationId);
if (additionalHeaders != null && !additionalHeaders.isEmpty()) {
    for (Entry<String, String> header : additionalHeaders.entrySet()) {
         httpGet.setHeader(header.getKey(), header.getValue());
    }
}
HttpResponse response = httpclient.execute(httpGet);
```

6. Save the changes.

3 Deploying a Secure Parameter Artifact

Use the secure parameter artifact to deploy confidential data for custom adapters.

Prerequisites

In the Node Explorer you have selected a tenant, in the context menu you have selected *Deploy Artifacts*, and you have specified *Secure Parameter* as the artifact type.

Context

You can store secure data like passwords in a secure store and use an alias name to access this data in an integration flow. The secure parameter artifact contains this alias name and confidential data.

You can deploy the artifact on the tenant by following the procedure below.

Procedure

1. Specify the attributes on the wizard page.

Option	Description
Name	Provide a name for the artifact. The artifact name is used as an alias for the confidential data assigned by this parameter.
Description	Provide a more detailed description of the artifact.
Secure Parameter	Enter the confidential value of the attribute.
Repeat Secure Parameter	Repeat the confidential value of the attribute.

2. Choose *Finish*.

> **i Note**
>
> If correct alias is configured in the integration flow, then the runtime framework passes the secured artifact value to the component at runtime from the secure store.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

www.sap.com/contactsap

THE BEST RUN SAP