

Browse the Book

SAP Cloud Platform Integration provides the option to access integration-related artifacts based on an application programming interface (API). The available APIs are described in detail in Chapter 9. This chapter also shows you how you can use Cloud Integration APIs together with SAP Cloud Platform API Management, a dedicated service of SAP Cloud Platform that helps you to develop, manage, and publish your own APIs.

-  **"Application Programming Interfaces"**
-  **Contents**
-  **Index**
-  **The Authors**

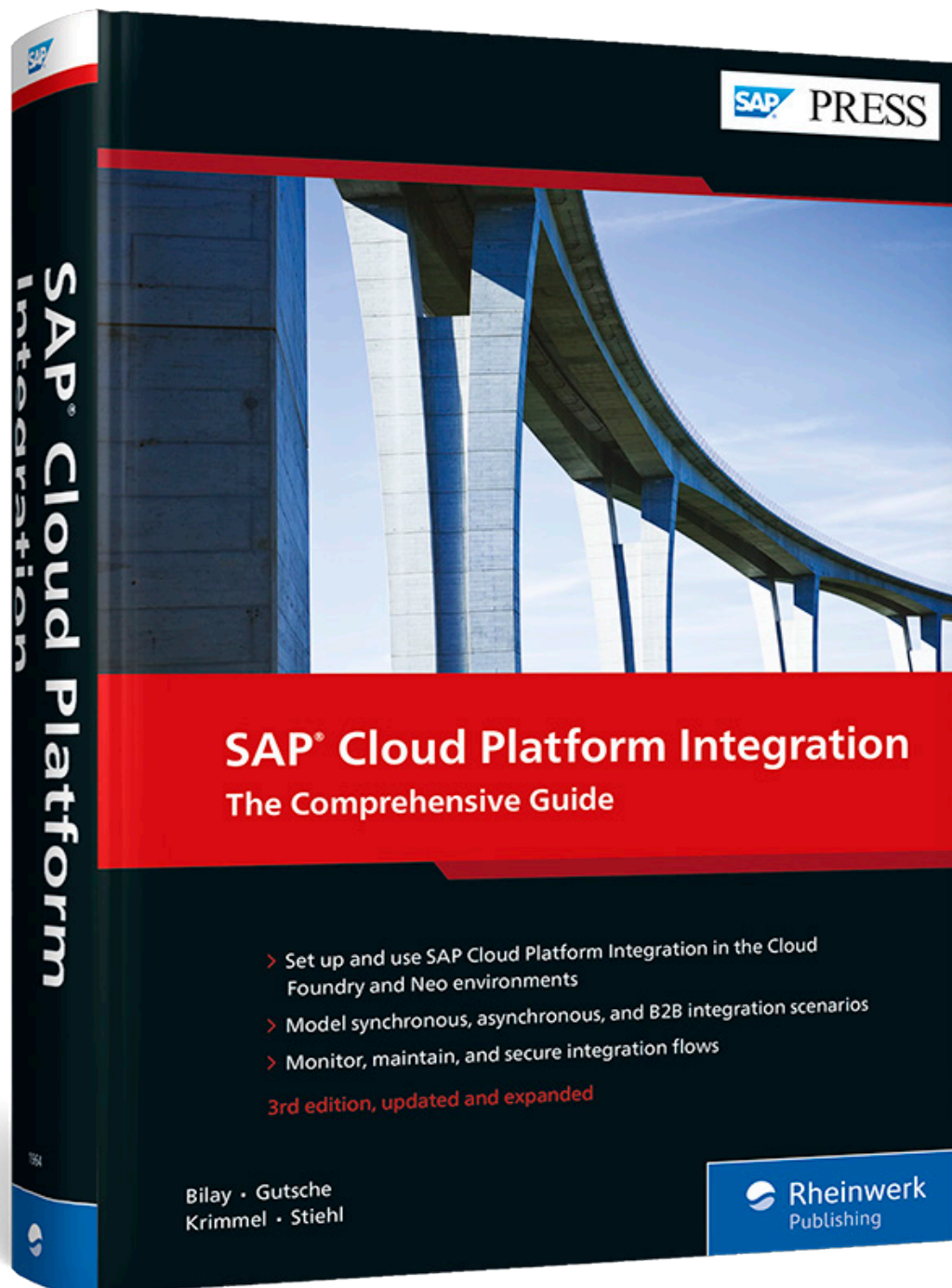
John Mutumba Bilay, Peter Gutsche, Mandy Krimmel, Volker Stiehl

SAP Cloud Platform Integration: The Comprehensive Guide

906 Pages, 2020, \$89.95

ISBN 978-1-4932-1964-3

 www.sap-press.com/5077



Chapter 9

Application Programming Interfaces

SAP enables the consumption and provision of application programming interfaces (APIs) to expose different functionalities to the outside world. This chapter dives into the specifics of using APIs in the context of SAP Cloud Platform Integration, presents available features, and explores the APIs currently available for customers.

In today's connected world, almost everything is a click away. From traditional desktops and mobile phones to connected devices like smartwatches, you can purchase goods, write articles, and book flight tickets. But how does data move from application A to B when, for instance, booking a flight or a car? The hidden enablers in most cases are known as application programming interfaces (APIs).

An API within the context of integration is an interface through which data exchange is made possible between applications. APIs are used to expose functionalities (or programmable interfaces) to the outside world through a service path or URL.

Simply put, an API takes a request from the caller, performs a task on a server application, and returns a response to the caller. The application providing the API is known as a *service provider*, and the application or user using the API is generally called a *service consumer*.

In this chapter, we'll introduce you to the Java and OData APIs provided by SAP Cloud Platform Integration and explain how to use them. This chapter further explores SAP Cloud Platform API Management and how to use it together with SAP Cloud Platform Integration.

9.1 Introduction

Much can be said about APIs—a topic that deserves its own book. This descriptive introduction is only intended to provide you a brief overview.

If you're familiar with the topic of integration, you might be wondering what the difference is between an API and a traditional web service. Table 9.1 points out some differences between an API and a web service.

Aspects	Web Service	API
Network	Needs a network connection for its operation	Can also operate offline
Protocols	Simple Object Access Protocol (SOAP), Representational State Transfer (REST), and XML-RPC. XML-RPC enables the calling of remote procedure using XML as the encoding and HTTP as the transport protocol.	SOAP and REST but can also communicate via cURL
Exposed via	XML over HTTP	Java Archive (JAR), Dynamic Link Library (DLL), XML, or JavaScript Object Notation (JSON) over HTTP

Table 9.1 Comparing Web Services to APIs

Furthermore, besides the differences listed in Table 9.1, all web services can be considered APIs, but not all APIs can be considered web services.

A *web service* is a type of API that almost always operates over HTTP (hence the “web” in the name). However, some protocols like SOAP (Simple Object Access Protocol) can use alternate transports, for example, Simple Mail Transfer Protocol (SMTP). The official World Wide Web Consortium (W3C) definition specifies that web services don’t necessarily use HTTP, but this is almost always the case and is usually assumed unless mentioned otherwise.

On the other hand, one could argue that every bit of every function ever created—whether Dynamic Link Library (DLL), Java Archive (JAR), web service, or plain code—is an API. APIs can use any type of communication protocol and aren’t limited to HTTP like web services are.

Now that you have a good, high-level understanding of what APIs are, let’s explore the Java APIs currently provided by SAP Cloud Platform Integration.

9.2 Java APIs Provided by SAP Cloud Platform Integration

SAP Cloud Platform Integration provides some Java-based APIs to access and control the processing of messages on your tenant. At the time of this writing, you can use Groovy or JavaScript as the programming language to access these APIs. Accessing the functionality provided by these APIs in a script step or while creating a user-defined function (UDF) for mapping can be handy, which we’ll describe in detail in Section 9.3.

Furthermore, you can use APIs when developing custom SAP Cloud Platform Integration adapters using the Adapter Development Kit (ADK). Note, however, that for a custom adapter, Java is used as a development language. We discussed the ADK in Chapter 6, Section 6.9.

Existing APIs can be classified under the following categories:

- **Generic APIs**
Complete and parent set of APIs covering various features. These APIs are kept in the package `com.sap.it.api`. Table 9.2 provides a list of interfaces contained in this package.
- **Message APIs**
Provides APIs to access properties of a message. These APIs are kept in the package `com.sap.it.api.msg`. Table 9.2 provides a list of interfaces contained in this package.
- **Script APIs**
Provides APIs to control scripts.
- **Mapping APIs**
Provides APIs to control mappings. These APIs are contained under the package `com.sap.it.api.mapping`. Table 9.2 lists a few commonly used packages and their corresponding interfaces.

Package	Interface	Description
<code>com.sap.it.api.msg</code>	<code>ExchangePropertyProvider</code>	Provides access to the properties of a message exchange.
<code>com.sap.it.api.msg</code>	<code>MessageSizeInformation</code>	Provides information about the size of a message.
<code>com.sap.it.api.mapping</code>	<code>MappingContext</code>	Mapping context object to be provided to mapping user-defined functions (UDFs).
<code>com.sap.it.api.mapping</code>	<code>Output</code>	Class used in advanced UDFs (execution type All values of Context or All values of a Queue) to return the result of a function.
<code>com.sap.it.api.mapping</code>	<code>ValueMappingApi</code>	Used to execute value mappings with the given parameters.
<code>com.sap.gateway.ip.core.customdev.util</code>	<code>Message</code>	Accesses the exchanged message. The API provides an extensive set of functionalities, including manipulating attachments; reading and changing payloads; and retrieving message properties such as size, header, and so on.

Table 9.2 API Packages and Interfaces

Package	Interface	Description
com.sap.it.script. logging	ILogger	Performs different operations on the logs (e.g., writing message logs).
com.sap.it.public. generic.api	ITApiException, Key- storeService, Secure- StoreService, UserCredential	Global API covering a wide range of functionalities, including access to key storage services, access to deployed user credentials, and access to exception objects.
com.sap.it.api.pd	PartnerDirectoryService	Performs different operations on Partner Directory parameter values, the partner IDs, the alternative partner IDs, and the authorized users of a partner. Using the Partner Directory was discussed in Chapter 7, Section 7.4.
com.sap.it.api.pd	BinaryData	Container for binary data relevant for Partner Directory binary parameters. Using the Partner Directory was discussed in Chapter 7, Section 7.4.

Table 9.2 API Packages and Interfaces (Cont.)

For a full list of interfaces and classes, refer to the JavaDocs at <http://s-prs.co/5077150>.

Note

Note that the summary of API packages listed in Table 9.2 relates to API version 2.12.0.

To illustrate the usage of the APIs, let’s next look at UDFs in a mapping.

9.3 Using the Java API in a User-Defined Function

Let’s say you need a message mapping to perform a complex transformation between a source and target message. In Chapter 4, Section 4.4, you learned how to work with mappings. Furthermore, imagine that none of the existing standard functions can fulfill the needed logic. Luckily, a user-defined function (UDF) can come to the rescue.

A UDF is a custom function built for special mapping needs that can’t be expressed by the predefined mapping functions in the mapping editor. In SAP Cloud Platform Integration, a UDF can be built using Groovy or JavaScript. A UDF generally uses the

mapping-related APIs listed in Table 9.2 to transform the messages. (To see an illustration of a UDF in use, go to the example in Chapter 4, Section 4.4.)

In this section, we’ve built an integration flow that consumes an external OData service. After invoking the OData service, we received the response shown in Figure 9.1.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header/>
  <soap:Body>
    <ns1:OrderShippingDetails_MT xmlns:ns1="http://hci.sap.com/demo">
      <orderNumber>10249</orderNumber>
      <customerName>Toms Spezialitäten</customerName>
      <shipCity>Münster</shipCity>
      <shipStreet>Luisenstr. 48</shipStreet>
      <shipPostalCode>44087</shipPostalCode>
      <shipCountry>DE</shipCountry>
      <shipDate>1996-07-10T00:00:00.000</shipDate>
    </ns1:OrderShippingDetails_MT>
  </soap:Body>
</soap:Envelope>
```

Figure 9.1 Response of the Integration Flow from Chapter 4

Let’s imagine that, with regard to the consumer application, you would prefer not to have the empty spaces in the element shipStreet between the street name and the house number. Furthermore, let’s say the consumer requires that the street field to be appended with the order number as a suffix, and the entire field should be returned in uppercase. Looking at the example shown in Figure 9.1, the value Luisenstr. 48 should be transformed into LUISENSTR.48_10249.

You can implement this requirement by using a combination of predefined functions in the mapping. However, for the sake of illustration, let’s try to achieve this requirement using a UDF by following these steps:

- 1. On the **Design** page of SAP Cloud Platform Integration Web UI, navigate to the concerned package and open the integration flow.
- 2. Click the **Edit** button.
- 3. Select the **Message Mapping** step to be enhanced with the UDF, as shown in Figure 9.2.

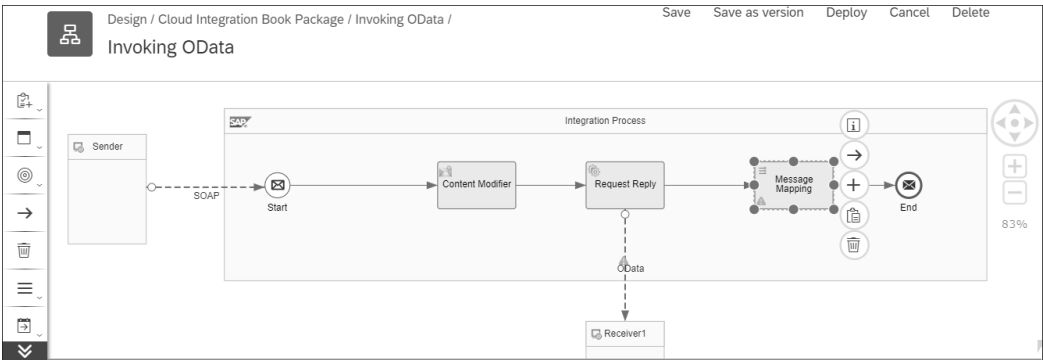


Figure 9.2 Selecting the Message Mapping Step to Be Enhanced with a UDF

- Under the **Processing** tab, click on the `/ODate2XML.mmap` link, as shown in Figure 9.2, to open the mapping editor.
- Select an element on the target message structure (e.g., the `shipStreet` field shown in Figure 9.3). Notice that a section called **Functions** appears in the bottom-left corner of the mapping editor, as shown in Figure 9.3.

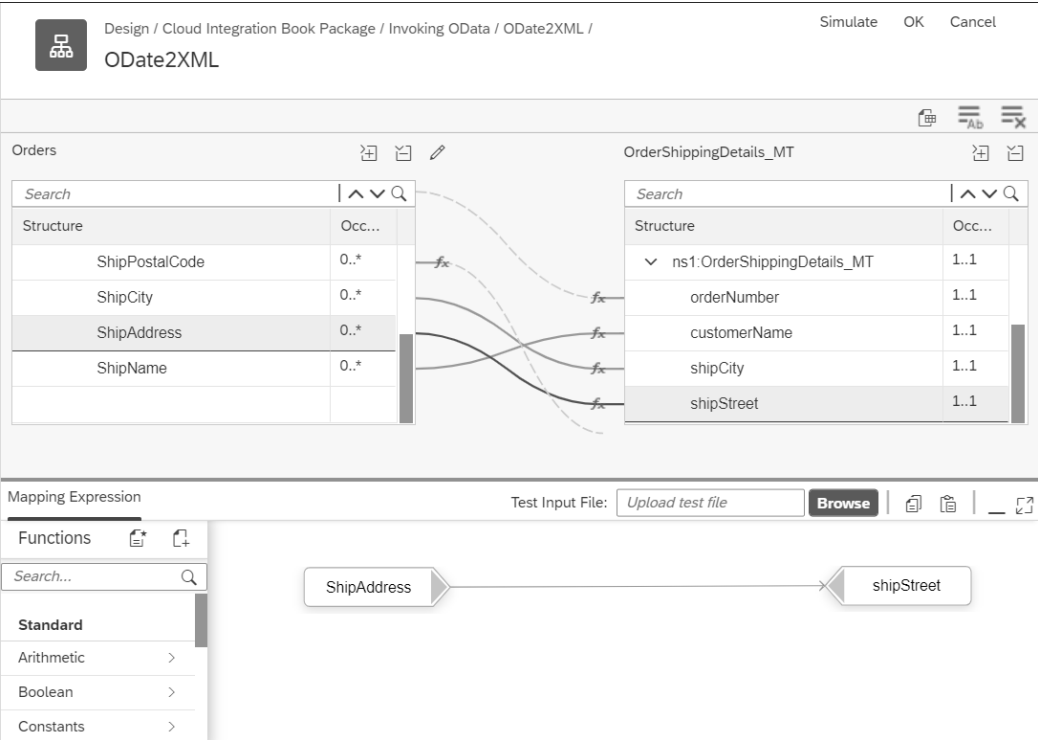


Figure 9.3 Mapping Editor with the Details of the Mapping to Be Enhanced


- Let's now create a new UDF by clicking the **Create** icon  next to the **Functions** box in the mapping editor.
- Specify a name for the UDF to be created, as shown in Figure 9.4.



Figure 9.4 Providing a Name for the UDF

You're redirected to the UDF editor, which will contain some standard code, as shown in Figure 9.5.

Note

As shown at the top of Figure 9.5, the mapping API has been imported by the statement `import com.sap.it.api.mapping.*;`, which is the same package described earlier in Table 9.2.

Note that Figure 9.5 also shows some sample source code to showcase what is possible. The sample code is included between the `/*` and `*/` characters.



Figure 9.5 Standard Groovy Script Code Provided When Creating a UDF

- Rename the Groovy method (e.g., "streetFormatterFunc") and adapt the source code to suit your needs, for example, with the code shown in Figure 9.6. In line 5, this code removes all extra spaces and converts the result to uppercase. Then, line 6 of the code shown in Figure 9.6 indicates that the code provided is using the Java API to retrieve the `OrderNo` from the message header. Furthermore, line 8 concatenates the input with the retrieved `OrderNo` from the message header.



Figure 9.6 Code Adapted to Remove Empty Spaces, Retrieve Message Header Attributes, and Perform a Concatenation

- Click the **OK** button in the top-right corner of the screen shown in Figure 9.6.

10. Return to the mapping editor and find the newly created **StreetFormatter** UDF under the **Custom** section, as shown in Figure 9.7.

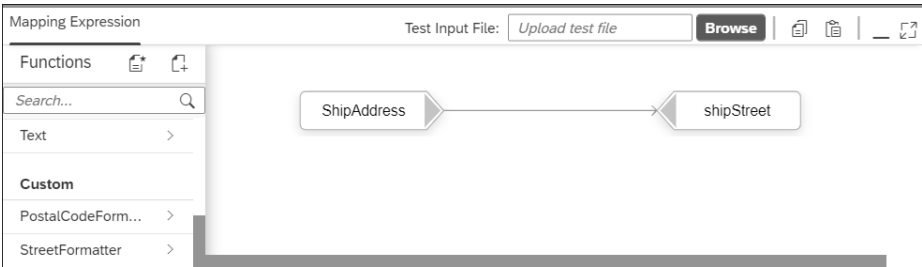


Figure 9.7 The Newly Created UDF Now Available in the Mapping

11. Click on the **StreetFormatter** UDF to see its method name. In our example, the method is called **streetFormatterFunc**, as shown in Figure 9.8. Note that the **streetFormatterFunc** comes from the name that we provided for the Groovy method shown earlier in Figure 9.6.
12. Let's now use our new function by dragging and dropping it between **ShipAddress** and **shipStreet**, as shown in Figure 9.8.

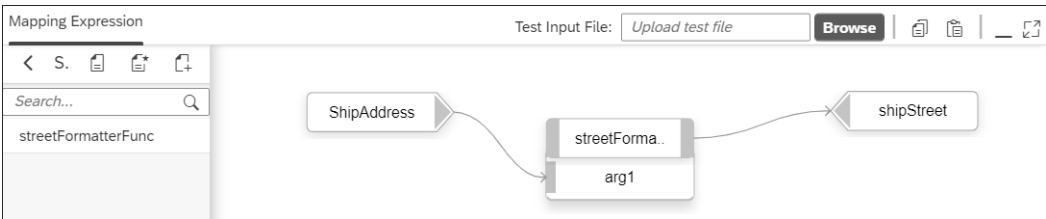


Figure 9.8 Inserting the UDF into Your Mapping Logic

13. Save and deploy the integration flow.
14. Test the service using SoapUI. You'll get a response in the field **shipStreet** without spaces, in uppercase, and suffixed with the order number, as shown in Figure 9.9.

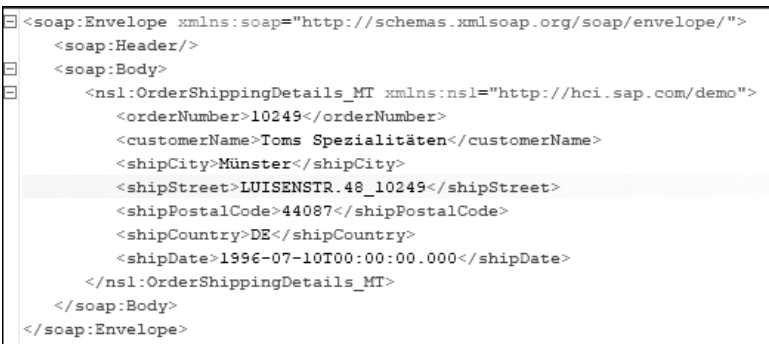


Figure 9.9 Response Returned by the Integration Flow after Adding the UDF

You now know how to create a UDF that uses SAP Cloud Platform Integration Java APIs to perform transformation logic in a mapping. Let's now move to the next section, where we'll show you how to use a script step in SAP Cloud Platform Integration.

9.4 Using the Script Step

SAP Cloud Platform Integration provides a script step that enables you to write different custom scripts to perform a wide range of activities and utilize the Java APIs we explored earlier in Section 9.2. The scripting feature opens the door to your development imagination so you can do almost anything. Note, however, that scripting should be used with due diligence and caution to avoid unnecessary overheads and performance issues.

At the time of this writing, the script step supports Groovy and JavaScript. Groovy is a Java syntax-compatible, object-oriented programming language for Java platforms. To learn more about Groovy, go to <http://groovy-lang.org>.

JavaScript is a dynamic, weakly typed, prototype-based, and multiparadigm programming language for the web. Many websites use JavaScript. To learn more about JavaScript, go to www.w3schools.com/js.

Both languages are relatively easy to learn, and plenty of resources are available on the Internet that you can use as references.

SAP Cloud Platform Integration provides a Script API in a form of a JAR file. Using this JAR file, a Java developer can easily import a library into his or her development tool of preference and inspect the provided methods. The Script API JAR file can be downloaded from <https://tools.hana.ondemand.com/#cloudintegration>. On that page, search for the **Using Script API** section, as shown in Figure 9.10. Then, click on the **Download** link to save the JAR file to your local file system.

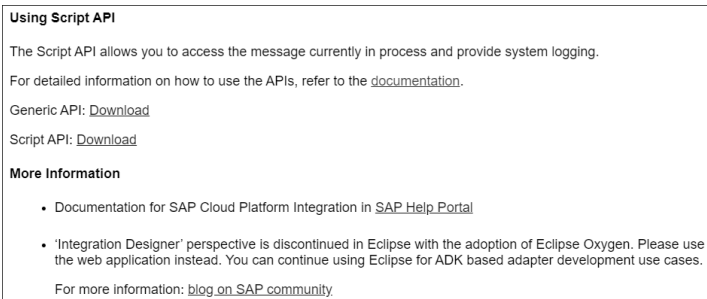


Figure 9.10 Downloading the Script API JAR File

At the time of this writing, the JAR file is called *cloud.integration.script.apis-1.36.1.jar*. To better illustrate the usage of the Script API, let's use a sample scenario in the next section.

9.4.1 Target Scenario

Let’s reuse the example from Chapter 4, Section 4.3, to illustrate the use of the script step. In that example, we invoked an external OData service from our integration flow. Figure 9.11 shows the integration flow that we built for it.

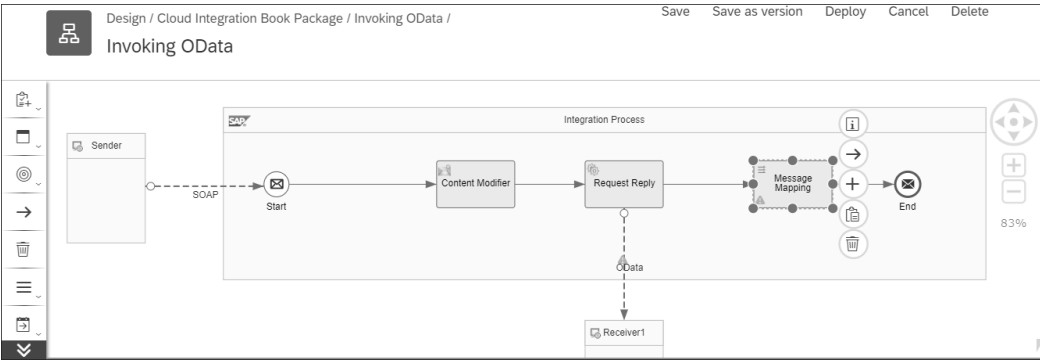


Figure 9.11 Invoking an OData Service

Imagine that the organization providing this integration flow is currently running a lottery on every incoming message. The sender of one randomly selected message will win a prize. As an integration developer, you’ve been asked to change the integration flow by generating a random number to be associated with each call. This random number will be used by your organization later to pick the lucky winner.

Keeping the solution simple, you can create a script that generates a random number and uses the API to save this value as a message header.

9.4.2 Enhancing the Integration Flow

Let’s now change our integration flow. Note that we won’t describe every step because, at this point, you should already be familiar with editing integration flows. Follow these steps:

1. On the **Design** page of the SAP Cloud Platform Integration Web UI, navigate to the correct package and integration flow.
2. In the opened integration flow, click the **Edit** button.
3. Select a **Script** step from the palette, as shown in Figure 9.12. You’ll find this shape in the palette under **Message Transformers**.
4. Another menu appears from the palette with a choice of **JavaScript** or **GroovyScript**. Click on **GroovyScript**, as shown in Figure 9.13.
5. Place the shape in your integration flow, right after the **Start** icon.
6. The **Script Editor** automatically opens with some sample script, as shown in Figure 9.14. Note that this editor is the same tool we used when we created a UDF in Section 9.3.

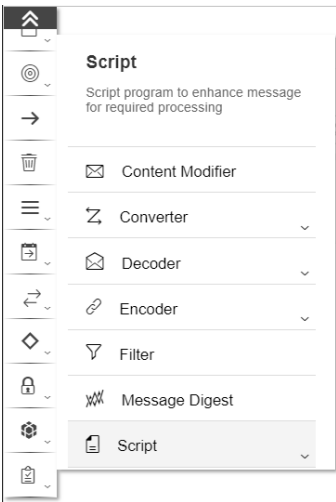


Figure 9.12 Adding the Script Step to the Integration Flow

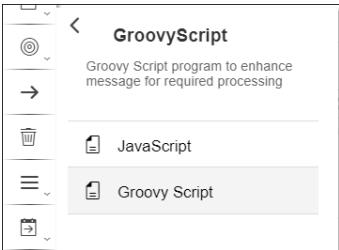


Figure 9.13 Selecting Groovy Script from the Palette

```
... / SampleIntegrationFlow / script1.groovy /
OK Cancel
script1.groovy

1  /*
2  The integration developer needs to create the method processData
3  This method takes Message object of package com.sap.gateway.ip.core.customdev.util
4  which includes helper methods useful for the content developer:
5  The methods available are:
6  public java.lang.Object getBody()
7  public void setBody(java.lang.Object exchangeBody)
8  public java.util.Map<java.lang.String,java.lang.Object> getHeaders()
9  public void setHeaders(java.util.Map<java.lang.String,java.lang.Object> exchangeHeaders)
10 public void setHeader(java.lang.String name, java.lang.Object value)
11 public java.util.Map<java.lang.String,java.lang.Object> getProperties()
12 public void setProperties(java.util.Map<java.lang.String,java.lang.Object> exchangeProperties)
13 public void setProperty(java.lang.String name, java.lang.Object value)
14 public java.util.List<com.sap.gateway.ip.core.customdev.util.SoapHeader> getSoapHeaders()
15 public void setSoapHeaders(java.util.List<com.sap.gateway.ip.core.customdev.util.SoapHeader>
16 soapHeaders)
17 public void clearSoapHeaders()
18 */
19 import com.sap.gateway.ip.core.customdev.util.Message;
20 import java.util.HashMap;
21 def Message processData(Message message) {
22 //Body
23 def body = message.getBody();
24 message.setBody(body + "Body is modified");
25 //Headers
26 def map = message.getHeaders();
27 def value = map.get("oldHeader");
28 message.setHeader("oldHeader", value + "modified");
29 message.setHeader("newHeader", "newHeader");
30 //Properties
31 map = message.getProperties();
32 value = map.get("oldProperty");
33 message.setProperty("oldProperty", value + "modified");
34 message.setProperty("newProperty", "newProperty");
35 return message;
}
```

Figure 9.14 Sample Code Included in the Groovy Script Editor

7. Change the processData method part of the script shown in Figure 9.14 to adapt the script to meet your requirements. In our case, we used the code shown in Figure 9.15. Note that the processData method takes a Message object as input and also returns a

Message object as an output. If you're a programmer, the script shown in Figure 9.15 is self-explanatory. The code generates a random number between 0 and 1000. The generated random number is then added as a message header named `LuckyNumber`. The code also includes comments in plain English for those less familiar with Groovy. The Groovy script shown in Figure 9.15 is included with the book's downloads at www.sap-press.com/5077.

Design / Cloud Integration Book Package / Invoking OData / script2.groovy / OK Cancel

script2.groovy

```
1  /*
2  The integration developer needs to create the method processData
3  This method takes Message object of package com.sap.gateway.ip.core.customdev.util
4  which includes helper methods useful for the content developer:
5  The methods available are:
6  public java.lang.Object getBody()
7  public void setBody(java.lang.Object exchangeBody)
8  public java.util.Map<java.lang.String,java.lang.Object> getHeaders()
9  public void setHeaders(java.util.Map<java.lang.String,java.lang.Object> exchangeHeaders)
10 public void setHeader(java.lang.String name, java.lang.Object value)
11 public java.util.Map<java.lang.String,java.lang.Object> getProperties()
12 public void setProperties(java.util.Map<java.lang.String,java.lang.Object> exchangeProperties)
13 public void setProperty(java.lang.String name, java.lang.Object value)
14 */
15 import com.sap.gateway.ip.core.customdev.util.Message;
16 import java.util.HashMap;
17 def Message processData(Message message) {
18     //Generate a random Number
19     int max = 1000;
20     int min = 0;
21     Random r = new Random();
22     def value = r.nextInt((max - min) + 1) + min;
23     //Properties
24     map = message.getProperties();
25     message.setHeader("LuckyNumber", value);
26
27     return message;
28 }
29 }
```

Figure 9.15 Final Result of the Groovy Script

8. Click the **OK** button in the top-right corner of Figure 9.15 to return to the integration flow. Figure 9.16 shows the final look of the integration flow extended with the **Script** step.

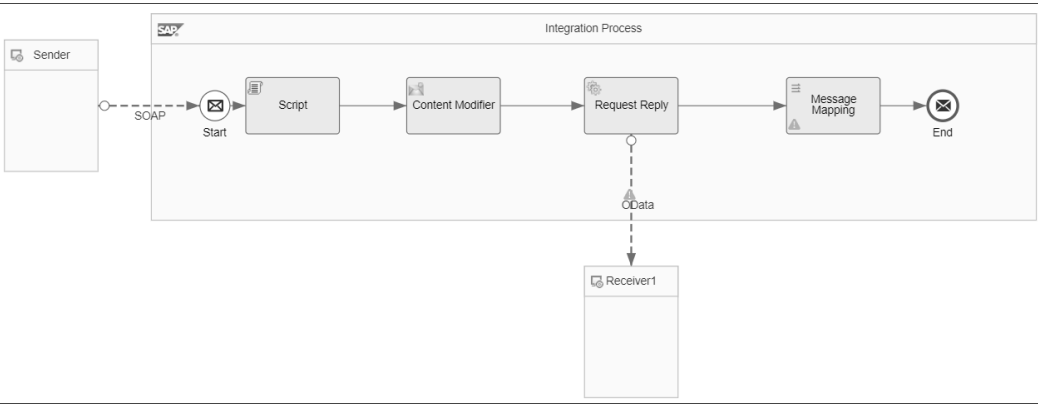


Figure 9.16 Enhanced Overview of the Integration Flow

9. Save and deploy the integration flow.

You're now ready to send a message via SoapUI. After you've triggered the message, the randomly generated number is returned in the header of the response message. Notice the `LuckyNumber` in the header section of the response shown in Figure 9.17 and Figure 9.18. This result is exactly what we asked the script to do (see line 15 of the code shown in Figure 9.16).

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:demo="http://cpi.sap.com/demo">
  <soapenv:Header/>
  <soapenv:Body>
    <demo:OrderNumber_MT>
      <orderNumber>10249</orderNumber>
    </demo:OrderNumber_MT>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 9.17 Returned Response Headers A

Body		Cookies (4)	Headers (21)	Test Results	Status: 200 OK	Time: 3.48s	Size: 1.04 KB	Save Response
		KEY		VALUE				
		Set-Cookie		JSESSIONID=11696AE7D64F68045F9AD20892401CDF; Path=/; Secure; HttpOnly				
		Address		http://services.odata.org/Northwind				
		AuthenticationType		None				
		clientSidePageSize		200				
		ComponentContentType		xml				
		destinationAlias		IGNORE				
		destinationManagerImpl		com.sap.it.rt.adapter.odata.destination.HCIDestinationManagerImpl@6858bc27				
		HttpStatusCodes		OK				
		LuckyNumber		516				

Figure 9.18 Returned Response Headers B

Congratulations! You can now use the **Script** step to perform different tasks using the APIs. Let's now move to discuss the OData API.

9.5 OData API

Besides the Java APIs, SAP Cloud Platform Integration also allows you to access various aspects of the platform using an OData API. These APIs are Representational State Transfer (REST) APIs that use the Open Data Protocol (OData) as a technical protocol. As a result, these APIs use the well-known HTTP methods of GET, POST, PUT, FETCH, and DELETE. The concept of OData is not completely new; you already came across OData in Chapter 7, Section 7.4.

Note

Note that, at the time of this writing, OData specification version 2.0 is supported by the SAP Cloud Platform Integration OData APIs. To read more about the OData V2 specification, go to www.odata.org/documentation/odata-version-2-0.

The OData APIs can be accessed using an HTTP URL in the following format:

https://<host>/api/v1/<resource>?\$<property1>=<property1_value>&\$<property2>=<property2_value>

In this URL, note the following:

- **<host>**
Represents the URL address of the service instance in Cloud Foundry, also called the SAP Cloud Platform Integration Management Host Address. We introduced this service in Chapter 2, Section 2.1.3. In the Neo environment, you'll need to use the URL address of the tenant management node.
- **<resource>**
Represents the path of the entity types to be called. Some entity resources are listed later in Table 9.5. For example, you can use the resource `MessageProcessingLogs` to address the message processing log (MPL).
- **<property1>**
Represents the name of the property to be queried. For example, you can use the property `count` to return the total number of MPLs. This property is always prefixed by a dollar sign (\$). Note that the property field is optional. Furthermore, you're can add as many properties as you need by using an ampersand (&) between them.

OData APIs are protected by OAuth in Cloud Foundry, whereas in the Neo environment these same OData APIs are protected by basic authentication (user name and password) and require the API client to enable HTTP cookies. Furthermore, to use an API, you must have the correct role collection assigned to your user. Table 9.3 lists the role collections required for the various API actions in Cloud Foundry. Similarly, Table 9.4 lists the authorization groups required for different API actions in the Neo environment.

Role Collection	Description
PI_Business_Expert	Enables a business expert to perform tasks such as monitoring integration flows and monitoring message content stored in temporary storage
PI_Administrator	Enables the tenant administrator to perform administrative tasks on the tenant cluster, for example, deploying security content and integration flows
PI_Integration_Developer	Enables an integration developer to display, download, and deploy artifacts (e.g., integration flows)
PI_Read_Only	Enables a user to display integration content and to monitor messages

Table 9.3 Required Authorization Groups for Cloud Foundry

Authorization Group	Description
AuthGroup.Administrator or AuthGroup.IntegrationDeveloper	Ability to display message overview
AuthGroup.IntegrationDeveloper	Undeploy integration content
AuthGroup.BusinessExpert	Download a message

Table 9.4 Required Authorization Groups for Neo

SAP Cloud Platform Integration's OData APIs are structured around entity types or resources. Every entity type contains a number of properties. A property can also refer to another entity type, which means you can start with one entity type and navigate to another entity type. Therefore, you must fully understand the various entity types, their tasks, and their relationships with each other. Table 9.5 lists all available entity types and describes their use.

Entity Types	Task Description
MessageProcessingLog	Reads an MPL
MessagePropocessingLogCustomHeaderProperty	Reads custom header properties of the MPL
MessageProcessingLogErrorInformation	Reads error information for a message
MessageProcessingLogAdapterAttribute	Reads adapter-specific attributes
MessageProcessingLogAttachment	Reads an MPL attachment
MessageStoreEntry	Reads a message from the message store
MessageStoreEntryProperty	Reads a header property of a message from the message store
MessageStoreEntryAttachment	Reads an attachment of a message from the message store
MessageStoreEntryAttachmentProperties	Reads properties of message attachments from the message store
IntegrationRuntimeArtifact	Reads properties of deployed integration content
PartnerDirectory	Accesses the Partner Directory, creates entries, and helps manage them
LogFile	Accesses all current (nonarchived) log files
LogFileArchives	Accesses all archived log files

Table 9.5 Entity Types and Their Tasks

To illustrate how entity types relate to each other, Figure 9.19 shows an entity model around the MessageProcessingLogs entity type to help you better grasp how to use the APIs related to the monitoring of message flows.

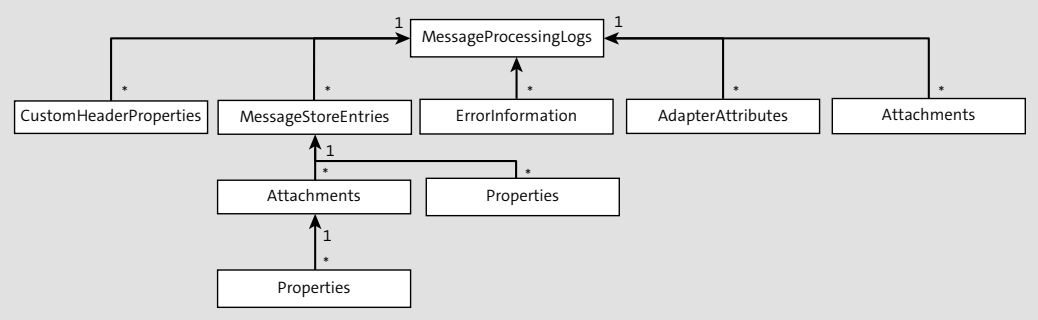


Figure 9.19 Entity Model Diagram for MessageProcessingLogs

As specified by the SAP documentation (shown in Figure 9.19), the MessageProcessingLogs entity contains several subentities, including CustomHeaderProperties, MessageStoreEntries, ErrorInformation, AdapterAttributes, and Attachments. Descriptions of each subentity’s function are listed in Table 9.5.

Additionally, the OData APIs provided by SAP Cloud Platform Integration make use of common query options. These query options can be used on different entity types to perform specific actions on them. However, not all options are supported by each entity type. Table 9.6 lists some common query options.

Option	Description
\$filter	Retrieves a set of entries based on the resource entity and the filter expression used in the Uniform Resource Identifier (URI)
\$metadata	Retrieves the data model and structure of all resources
\$select	Retrieves a subset of information on the entities identified by the resource path section of the URI
\$top	Returns a subset of <i>n</i> top records from the resource used in the URI
\$count	Returns the number of entries that matches the resource specified in the URI or the filter-specified criteria
\$inlinecount	Indicates that the response contains a count of the number of entries in the collection of records identified by the resource path section of the URI
\$value	Retrieves specific values of an entity resource specified by a Global Unique Identifier (GUID)

Table 9.6 Commonly Used Query Options

Option	Description
\$skip	Skips <i>n</i> records in the collection returned according to the resource path section of the URI
\$expand	Retrieves related and correlated entities for a given navigation property in line with the entities being retrieved
\$orderby	Specifies the sorting of the returned collection by one or more values

Table 9.6 Commonly Used Query Options (Cont.)

Note

The OData APIs provided by SAP Cloud Platform Integration limit the number of entries in a response to a maximum of 1,000 entries for each call. This limitation protects against negative impact in the performance of the SAP Cloud Platform Integration runtime environment and potential problems from queries returning huge amounts of data.

Queries with more than 1,000 entries are capped, and a **Next** link element is added to the response, which can be used to initiate the return of the additional entries.

Later in this chapter, we’ll explore APIs and entities related to the following aspects:

- Monitoring MPLs
- Deployed integration content
- Log files
- Message store
- Security material
- Partner Directory

These APIs can easily be tested and explored using the SAP API Business Hub, which we’ll discuss next.

9.5.1 SAP API Business Hub

The available OData APIs are exposed and documented in the SAP API Business Hub, which represents the central catalog of all SAP-provided and partner-developed APIs for developers to build sample apps, extensions, and open integrations with SAP. The SAP API Business Hub landing page can be found at <https://api.sap.com>, as shown in Figure 9.20.

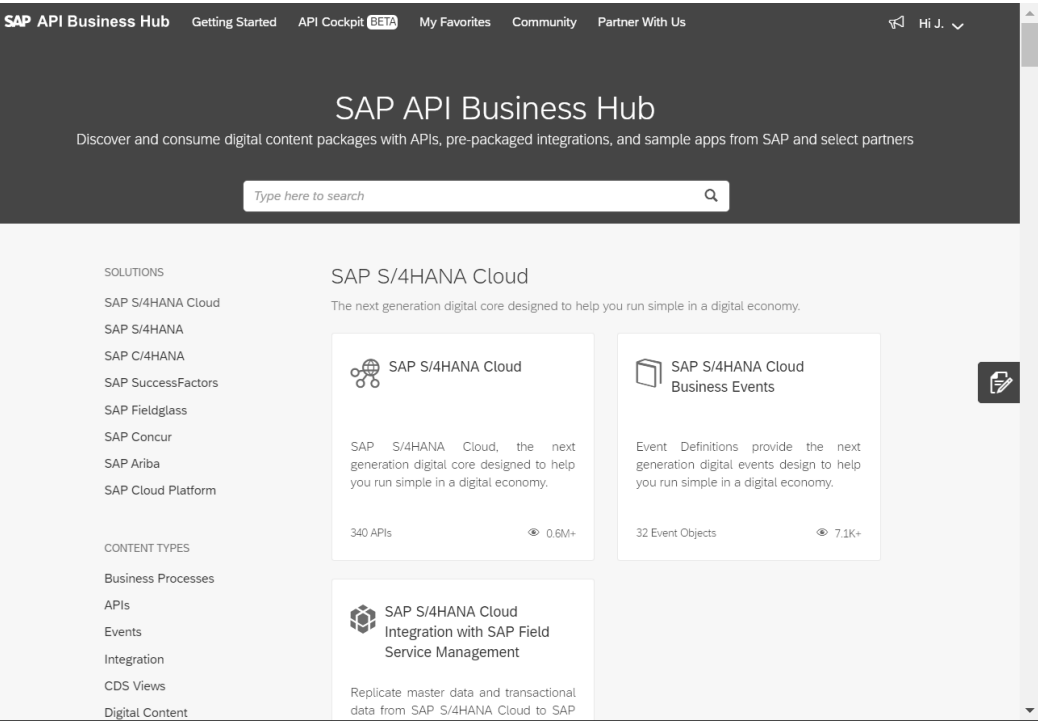


Figure 9.20 SAP API Business Hub Landing Page

Click on **APIs** on the left to open a view similar to the view shown in Figure 9.21. From this page, click on the **View more** link to move to another page where you can select different packages, as shown in Figure 9.22.

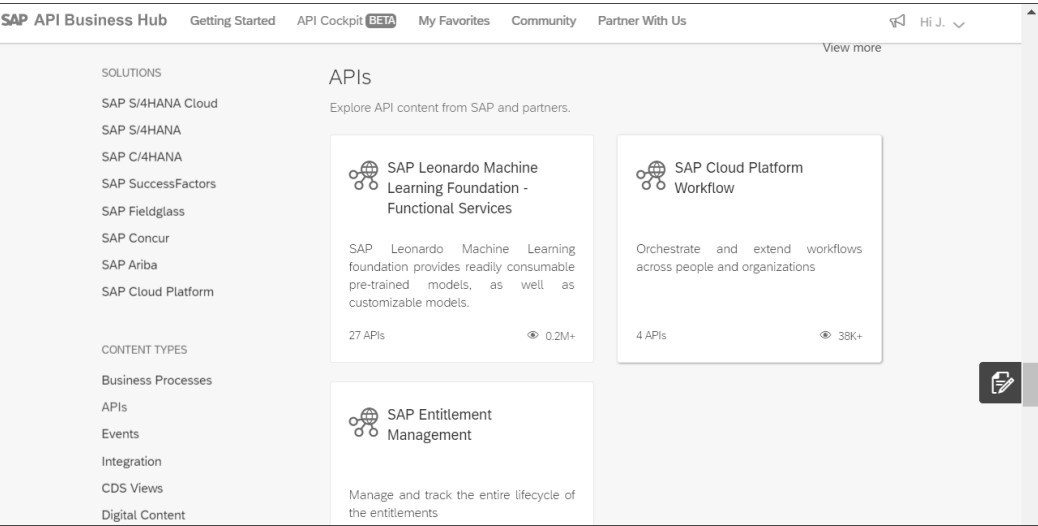


Figure 9.21 SAP API Business Hub Landing Page: APIs Section

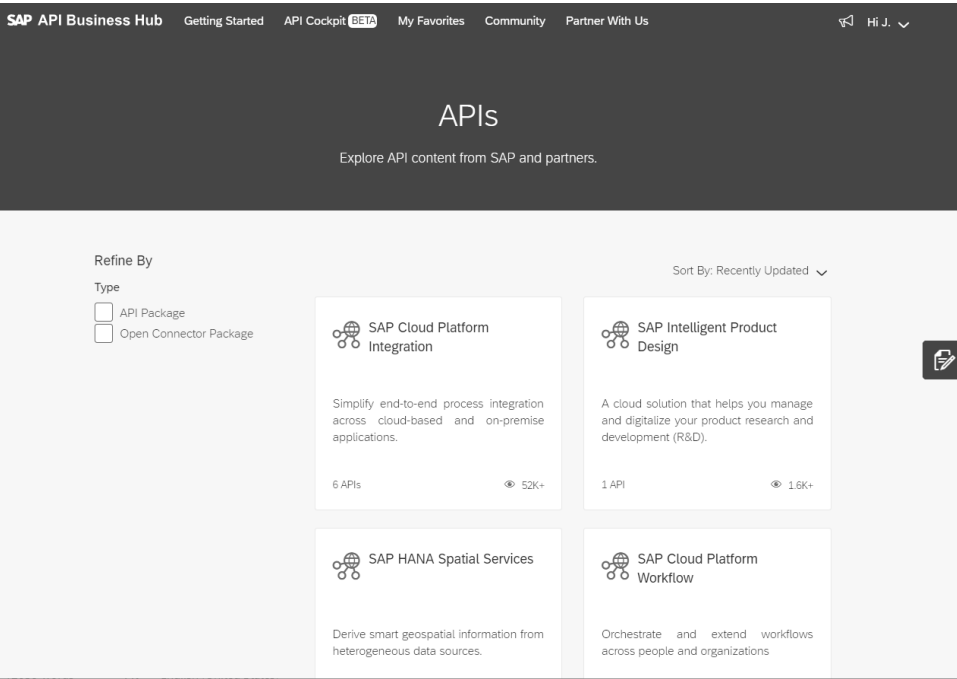


Figure 9.22 List of Available API Packages

Select the package called **SAP Cloud Platform Integration**. If you can't find this package, filter the list of packages by specifying a category, as shown in Figure 9.23.

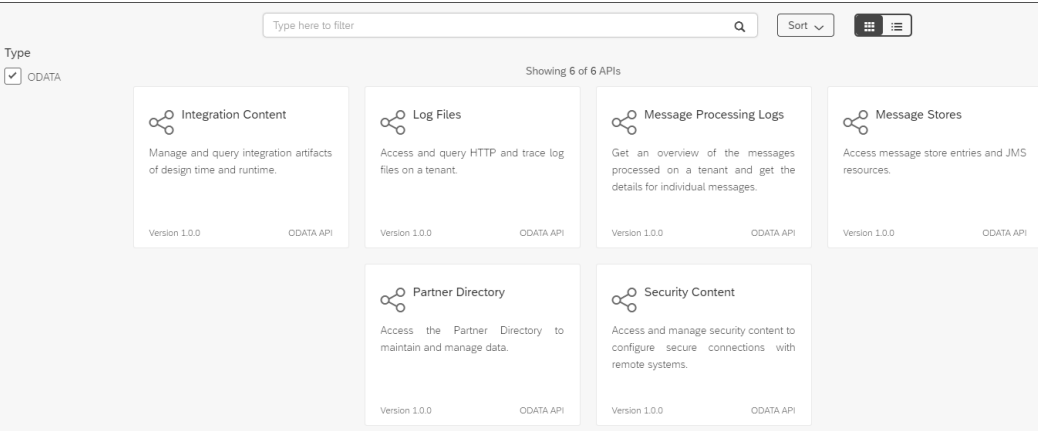


Figure 9.23 Artifacts of the OData APIs in the SAP Cloud Platform Integration Package

From the page shown in Figure 9.23, you can explore many APIs. Using the SAP API Business Hub, you can try out and test APIs directly without having to implement any code or using a third-party REST client such as Postman (www.getpostman.com). For APIs implementing the HTTP GET method, you can also use a simple browser.

The SAP API Business Hub has two different approaches to performing tests:

- An API sandbox
- Your SAP Cloud Platform Integration tenant

Each of these testing approaches will be explored next.

API Sandbox

If you don’t have access to an SAP Cloud Platform Integration tenant to use for testing, you can always use the API sandbox, which is provided by SAP. The API sandbox is filled with test data and presents a quick way to get a feel for the way the APIs operate. Note that only operations using the GET method are supported in the API sandbox. Operations needing write access are forbidden because you need to log on before you can call operations requiring write access in the API sandbox.

For instance, let’s use the Log Files API to illustrate testing using the API sandbox approach. Note that the Log Files API is not yet available in the Cloud Foundry environment at the time of this writing. But we can still use the Log Files API to illustrate how to test an API against a sandbox machine (which is still in a Neo environment). You’re already familiar with the Log Files API because, in Chapter 8, Section 8.5.2, we discussed how to access log files in the Web UI, on the **Monitor** page. For that, follow these steps:

1. Click on **Log Files** from the page shown earlier in Figure 9.23.
2. The resulting page lists all operations of the API. Note that the **API Endpoint** field is automatically assigned with a sandbox-related URL.
3. Assuming we want to test the **LogFileArchives** API, we’ll need to click on the **Show/Hide** link to the right of **LogFileArchives**, as shown in Figure 9.24.

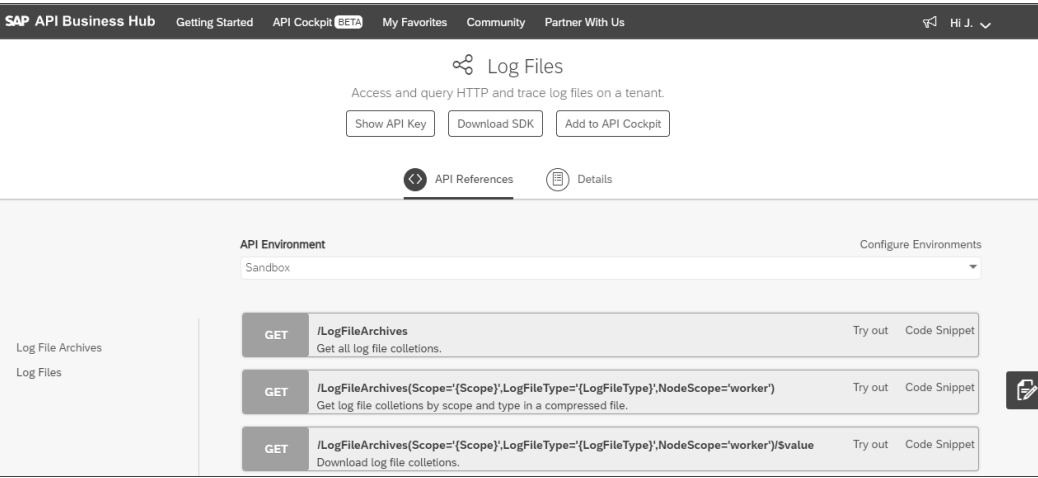


Figure 9.24 List of Entity Types Available for the Log Files API

4. From the resulting page, click the **GET** button to the left of **/LogFileArchives**.

5. A page similar to the page shown in Figure 9.25 will open. Click the **Try it out** button. Note that, if your API requires input parameters, these parameters will be listed on this page. You’ll need to fill in the required parameters at a minimum.

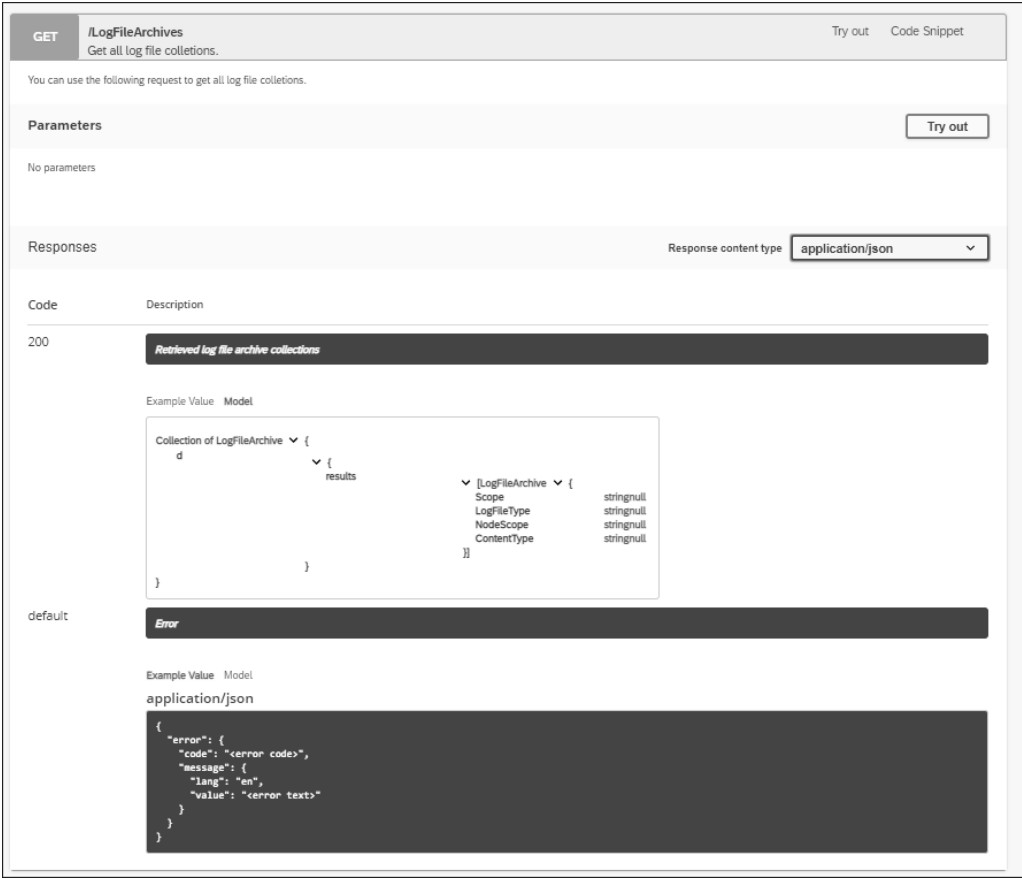


Figure 9.25 The SAP API Business Hub Try it Out Page

From the page shown in Figure 9.25, a number of attributes and functionalities are available, as listed in Table 9.7.

Properties	Description
Description	Describes the operation.
Parameters	Used to add the header or query parameters for the request message of the API.
Response Messages	Documents the possible response messages, including the different HTTP status codes and example response messages. After calling the API, both response body and response header are also returned.

Table 9.7 Properties in the SAP API Business Hub’s Try It Out Page

6. After clicking on the **Try it out** button, a response header and body will be returned on the page. Table 9.8 lists the functionalities and features available in the page shown in Figure 9.26.

Functionality	Description
Code Snippet	Retrieve the documentation and code snippet describing how to invoke this API. Currently, snippets in the following languages and technologies are supported: JavaScript, Java, Swift, cURL, ABAP, and SAPUI5 (as shown in Figure 9.26). This snippet can serve as a head start for your implementation.
Download API	Download the API in one of the following formats: JavaScript Object Notation (JSON), YAML Ain't Markup Language (YAML), or Entity Data Model Designer (EDMX).
Show API Key	Use to retrieve the API key in the sandbox host URL to try out APIs.
Download SDK	Download a prepopulated Java Software Development Kit (SDK) for the API. Note that this download includes a full Java project that you can use.

Table 9.8 Functionalities in the SAP API Business Hub’s Try It Out Page



Figure 9.26 Code Snippet to Consume the API in Different Languages

Let’s look at how to configure the SAP API Business Hub to test against your own SAP Cloud Platform Integration Cloud Foundry tenant in next.

Your SAP Cloud Platform Integration Tenant for Cloud Foundry

As mentioned earlier, SAP API Business Hub also makes it possible to test against your tenant. For this approach, you’ll need to configure the SAP API Business Hub to point to your tenant by changing the API endpoint. Follow these steps:

1. Click on the **Configure Environments** link in the top-right corner of the page shown earlier in Figure 9.24.
2. If you’re not already logged in, the **Login Required** popup window, shown in Figure 9.27, will open. Click the **Login** button and provide your credentials. If you’re already logged in, proceed to step 4.

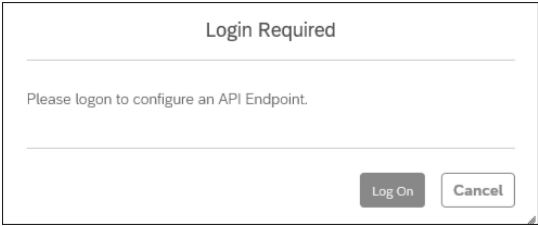


Figure 9.27 Login Page When Configuring the API Endpoint

3. You’re redirected back to the main page (shown earlier in Figure 9.24). Click on the **Configure Environments** link one more time. You’ll then presented with the popup window shown in Figure 9.28.

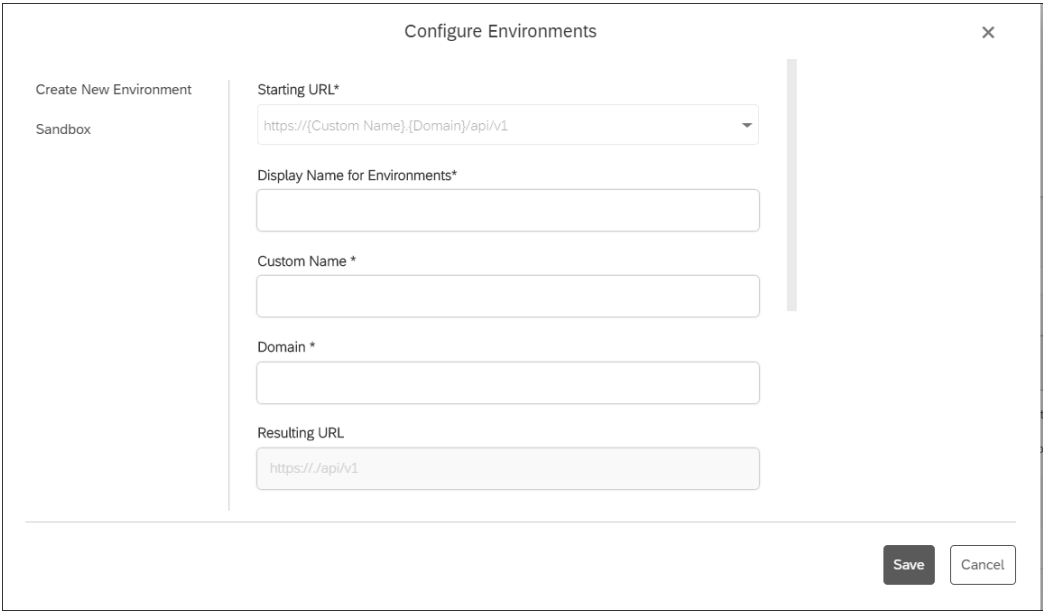


Figure 9.28 Filling in Environment Details for Your SAP Cloud Platform Integration Tenant

4. For the Cloud Foundry environment, you'll need to select **https://{Custom Name}/{Domain}/api/v1** from the **Starting URL** dropdown menu.
5. You'll then need to fill in the fields on the screen shown in Figure 9.28 according to the details listed in Table 9.9, which lists and describes all the relevant fields.

Column	Description
Display Name for Environment	Enter a human-readable alias name, for example, "Development," "Test," or "Acceptance."
Custom Name	Based on the URL of your Cloud Foundry tenant, the custom name is the part of the URL found between the double slashes (//) and the first dot (.).
Domain	Based on the URL of your Cloud Foundry tenant, the domain is the part of the URL found from the first dot (.) until the end of the URL.
Resulting URL	The resulting URL is automatically built for you. Ensure that this URL is the same as your Cloud Foundry tenant URL.
User name	The user name of your SAP Cloud Platform Integration tenant account.
Password	The password of your SAP Cloud Platform Integration tenant account.

Table 9.9 Attributes of the Configure API Endpoint Page for Cloud Foundry

Voilà! From this point, you can perform the API calls against your own tenant. As mentioned earlier, Cloud Foundry requires OAuth to connect to its APIs. (We'll discuss OAuth in more detail in Chapter 11, Section 11.4.3.) Additionally, you'll need to add specific authorization groups, listed earlier in Table 9.3, relevant to the operation to be performed. (These authorization groups were listed earlier in Table 9.3.) To enable OAuth, first, you must perform the following activities:

- Define a service instance, which represents the technical user to be used when calling the API later. Within the context of OAuth, a service instance represents an OAuth client.
- Define a service key for the API client, which contains the OAuth credential values including **client id**, **client secret**, **token URL**, and **url**.

You already came across these entities (the service instance and the service key) in Chapter 2 Section 2.3.3, when we defined authorization enabling a user to call and access an integration flow's endpoint. (We'll also further discuss these entities in Chapter 11, Section 11.4.4.) For now, let's discuss the steps required for each of the activities listed earlier.

To define a service instance, follow these steps:

1. From the SAP Cloud Platform cockpit, navigate to your subaccount and select the **Entitlements** menu item on the left, as shown in Figure 9.29. Then, verify that the **API Plan** under the **Process Integration on Runtime** service is available and enabled.

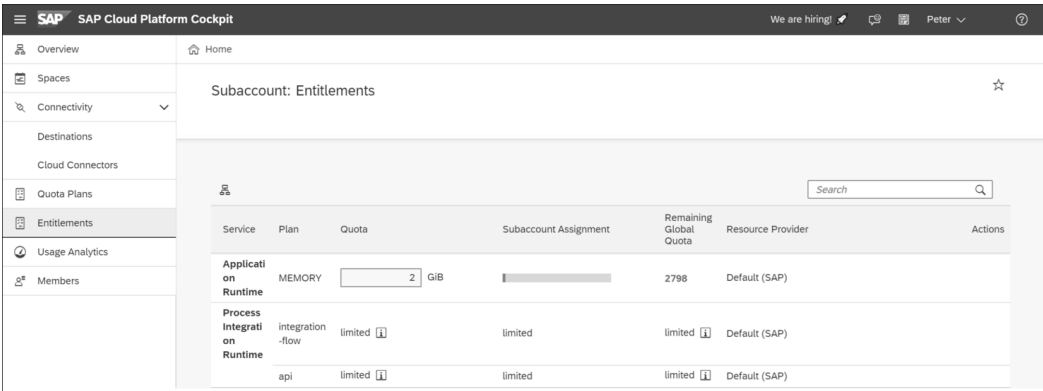


Figure 9.29 Checking That the API Plan Is Enabled

2. Then, navigate to the space management page by clicking on the **Spaces** menu item on the left, as shown in Figure 9.30.

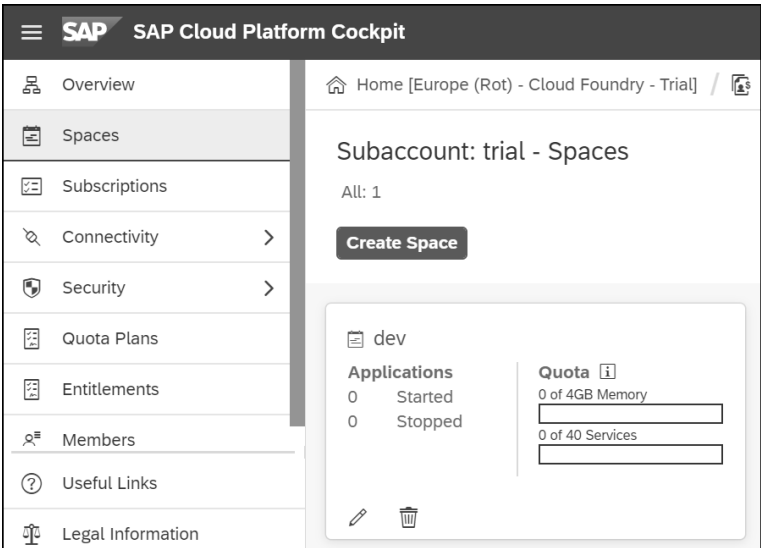


Figure 9.30 Selecting the Space to Hold the API Client

3. Select the space under which the API client should be created. For our example, as shown in Figure 9.30, you'll need to select the **dev** tile.
4. Under the **Services** menu item on the left, select **Service Marketplace**, as shown in Figure 9.31.

5. On the **Service Marketplace** page, search for the “process integration” keyword, as shown in Figure 9.31.

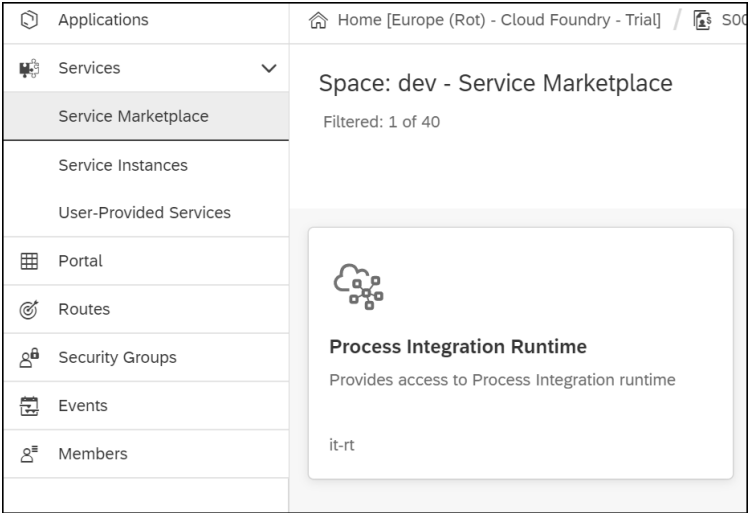


Figure 9.31 Selecting a Service from the Service Marketplace

6. On the returned results, select the **Process Integration Runtime** service.
7. Click on the **Instances** menu item on the left to view a list of existing instances, as shown in Figure 9.32.

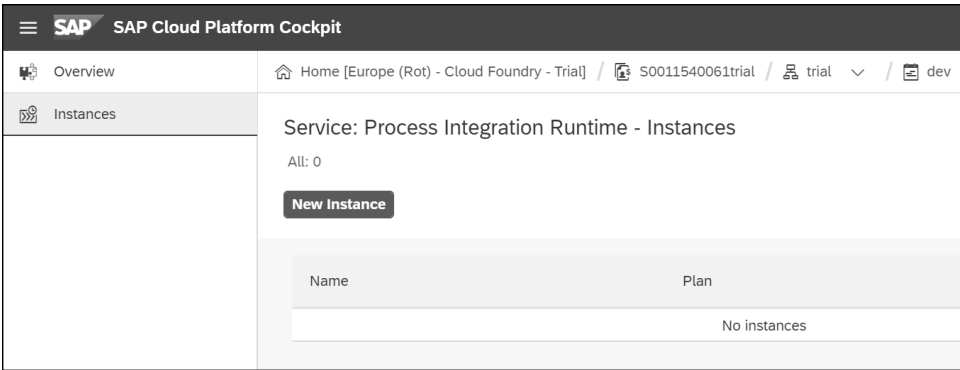


Figure 9.32 Creating a Process Integration Runtime Instance

8. To create an API client, click the **New instance** button, as shown in Figure 9.32. A new creation wizard will open.
9. On the next page, for the **Choose Service Plan** step, select **api** from the **Plan** drop-down list, as shown in Figure 9.33.
10. Then, click the **Next** button, located in the bottom-right corner of the page shown in Figure 9.33.

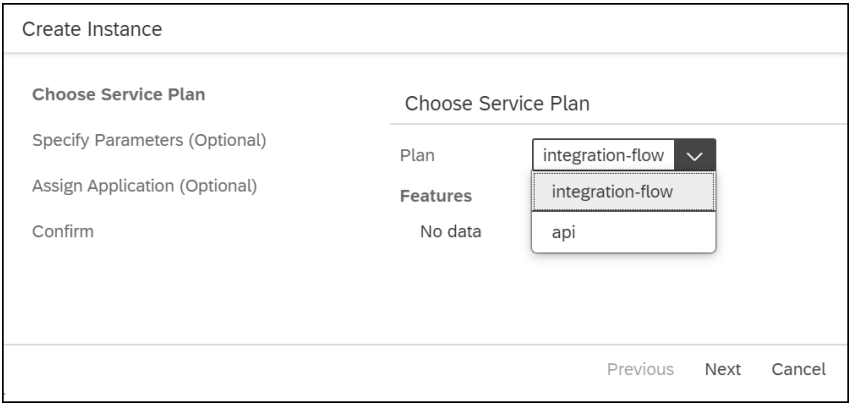


Figure 9.33 Selecting the API Service Plan

11. On the next page, you must enter a role assignment. Assuming that we’re trying to access the MPLs in our SAP Cloud Platform Integration on the Cloud Foundry environment, you’ll need to paste the sample JSON code shown in Listing 9.1.

```
{
  "roles": [
    "MonitoringDataRead"
  ]
}
```

Listing 9.1 Example JSON for Role Assignment

Note

A role assigned enables you to specify the type of action a user can perform on an API. The `MonitoringDataRead` role, used in Listing 9.1, enables a user to read monitoring data, including MPLs. For a full list of roles for both Neo and Cloud Foundry, refer to the documentation for SAP Cloud Platform Integration at https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud and search for “Tasks and Permissions.” To specify multiple roles for a user, separate the roles with a comma, for example, `MonitoringDataRead,WorkspaceArtifactsDeploy`.

12. Click the **Next** button, then **Next** again on the page that follows.
13. Then, enter an instance name of your choice and click **Finish**.

Now, create a service key by following these steps:

1. Select our newly created service instance to open a screen similar to the screen shown in Figure 9.34.
2. Click the **Create Service Key** button.
3. Provide a name for the service key and click **Save**.

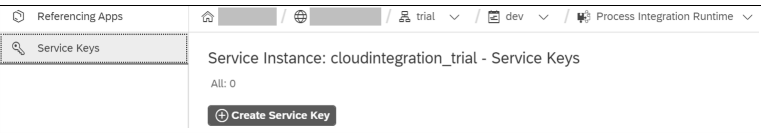


Figure 9.34 Creating a Service Key from a Service Instance

You’re then presented with parameters values of your service key. Note that the service key contains few security parameters, including `clientid`, `clientsecret`, `tokenurl`, and `url`, as shown in Figure 9.35. A description of each parameter field is provided in Table 9.10.

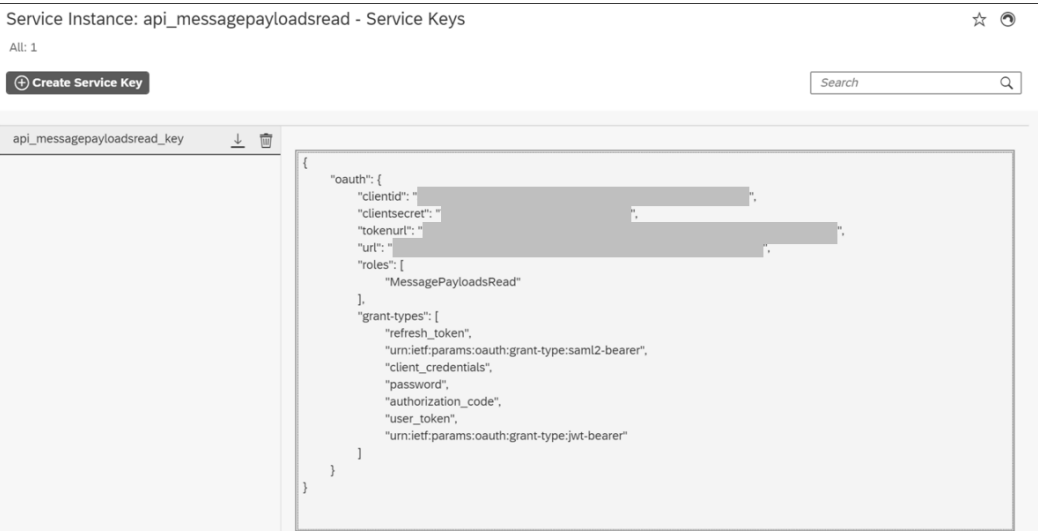


Figure 9.35 List of Generated Service Key Parameters to Be Used for OAuth Authentication

Parameter	Description
clientid	Represents the user name to be used by the API client to call the API.
clientsecret	Represents the password to be used by the API client to call the API.
tokenurl	The URL of the authorization server. This URL is responsible for issuing the OAuth token.
url	Represents the base URL of the OData API.

Table 9.10 Service Key Parameters

Voilà! You now have the details you need to perform an OAuth OData API call to your SAP Cloud Platform Integration in Cloud Foundry. Let’s now explore how to use the service key parameters we obtained earlier, as shown in Figure 9.35, to connect to an OData API using OAuth 2.0.

You can use any API client of your choice, but for illustration, we’ll use Postman (www.getpostman.com/), since you already installed Postman for a scenario in Chapter 7. You’ll first need to retrieve an OAuth token with the OAuth credentials by following these steps:

1. Perform an HTTP POST request on a request URL, which follows the format `https://<tokenurl>?grant_type=client_credentials`.
2. Select **Basic Auth** from the **Type** dropdown list, as shown in Figure 9.36.

Note

For the URL, the `<tokenurl>` can be retrieved from the screen shown in Figure 9.35. With the basic authentication, enter “clientid” in the **Username** field and “clientsecret” in the **Password** field. Both clientid and clientsecret values can be retrieved from the service key shown in Figure 9.35.

3. Trigger the call by clicking the **Send** button.

As a result of the call, an `access_token` is returned in the response. You’ll need to append the `access_token` to any subsequent OData API calls.

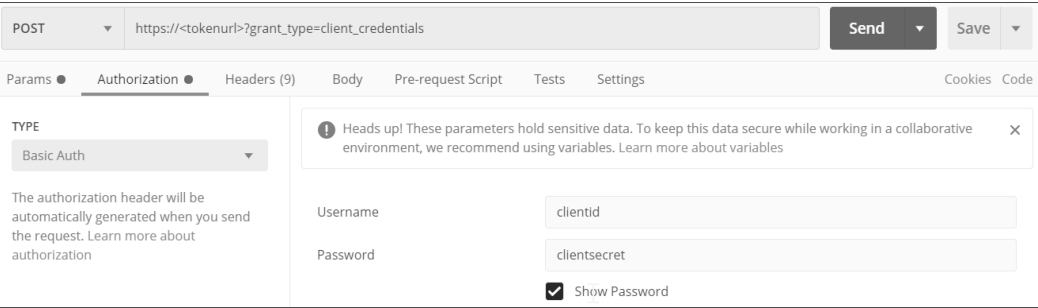


Figure 9.36 Postman Set Up to Retrieve the OAuth Token

Assuming that you intended to retrieve MPL data in your subsequent API call, you’ll need to use a URL in the following format:

`https://<tokenurl>/api/v1/MessageProcessingLogs?access_token=< access_token_value>`

Note that, for the `<access_token_value>`, use the `access_token` value returned from the previous call.

If you’re using SAP Cloud Platform Integration in the Neo environment, you can explore how to configure your tenant in the next section.

Your SAP Cloud Platform Integration Tenant for Neo

When using the Neo environment, you must select the region-specific host of your SAP Cloud Platform Integration tenant from the **Starting url** dropdown list, which is shown in Figure 9.37. Compare your tenant URL with one of the entries shown in Figure 9.37 to find the right match.

Then, maintain the **Display Name for Environment**, **Account Short Name**, and **SSL Host** fields and your user credentials. Refer to Table 9.11 for more details about these attributes.

Configure Environments

Create New Environment

Sandbox

Starting Url*

https://{Account Short Name}-tmn.{SSL Host}.api1.hana.ondemand.com/api/v1

Display Name for Environment*

Development

Account Short Name*

p001

SSL Host*

hci

Resulting URL

https://p001-tmn.hci.api1.hana.ondemand.com/api/v1

Authentication Type*

Basic Authentication

Username*

myuser

Password*

...

☐ Apply this environment to all APIs in this package that are not yet configured

Figure 9.37 Selecting the Region of Your Tenant for Neo

Column	Description
Display Name for Environment	Enter a human-readable alias name, for example, “Development,” “Test,” or “Acceptance.”

Table 9.11 Attributes of the Configure API Endpoint Page for Neo

Column	Description
Account Short Name	This value is your tenant ID, which can be found between the // and -tmn in your tenant URL. The tenant URL is always of the format: <i>https://{Account Short Name}-tmn.{SSLHost}.{Region}.hana.ondemand.com</i> .
SSL Host	This value can be retrieved from your tenant URL. The tenant URL is always of the format: <i>https://{Account Short Name}-tmn.{SSLHost}.{Region}.hana.ondemand.com</i> . Example: For {SSL Host}, use hci.
User name	The user name of your SAP Cloud Platform Integration tenant account.
Password	The password of your SAP Cloud Platform Integration tenant account.

Table 9.11 Attributes of the Configure API Endpoint Page for Neo (Cont.)

For those using the Neo environment, let’s now discuss the cross-site request forgery (CSRF) token handling in the next section. If you’re using Cloud Foundry, you can skip the next section.

9.5.2 Cross-Site Request Forgery Token Handling for Neo

Since APIs on the Neo environment use basic authentication, they are vulnerable for cross-site request forgery (CSRF) attacks. This type of security attack or malicious exploit occurs when unauthorized commands are transmitted from a user that the web application trusts. CSRF exploits the trust that a site has in a user’s browser. One example is a banking website that uses cookies to identify you in the future.

Within the context of APIs, thus, a CSRF attacker can execute an action on the target application via an API without the knowledge or permission of the consumer application. In general, CSRF attacks have the following characteristics:

- These attacks generally involve sites that rely on a user’s identity.
- They exploit the site’s trust in that identity.
- They trick the consumer application into sending HTTP requests to a target site.
- They involve HTTP requests that change application data.

To prevent CSRF attacks, some OData APIs provided by SAP Cloud Platform Integration require X-CSRF token validation. An X-CSRF token is mostly required for APIs that need permission to write and change objects via the POST, PUT, and DELETE HTTP operations. We don’t have the time to go into how X-CSRF tokens work, so for further reference, you can consult the many resources available online, including <http://s-prs.co/507754>.

When an API uses an X-CSRF token, calls made to the API without an X-CSRF token are rejected. As a result, you must retrieve an X-CSRF token first, before invoking such an API. Let’s now explore how you can fetch an X-CSRF token.

Let’s use Postman as our API client, since we already installed it back in Chapter 7. To retrieve the X-CSRF token, you’ll need to use the following endpoint:

- `https://<TMN-host>/api/v1` for Neo
- `https://<tokenurl>/api/v1` for Cloud Foundry

For the remaining sections in this chapter, whenever an OData endpoint is presented with `https://<tokenurl>/...`, you can assume that this represents a Cloud Foundry URL. The equivalent Neo URL can be constructed by replacing `<tokenurl>` with `<TMN-host>`.

In this context, `<TMN-host>` represents the tenant host of SAP Cloud Platform Integration. As shown in Figure 9.38, select **GET** as the HTTP method and specify the OData API endpoint. For Neo, you’ll also need to select **Basic Auth** and enter your credentials. For Cloud Foundry, you’ll need to select **No Auth** and extend the URL with the `access_token` query parameter, as discussed in Section 9.5.1.

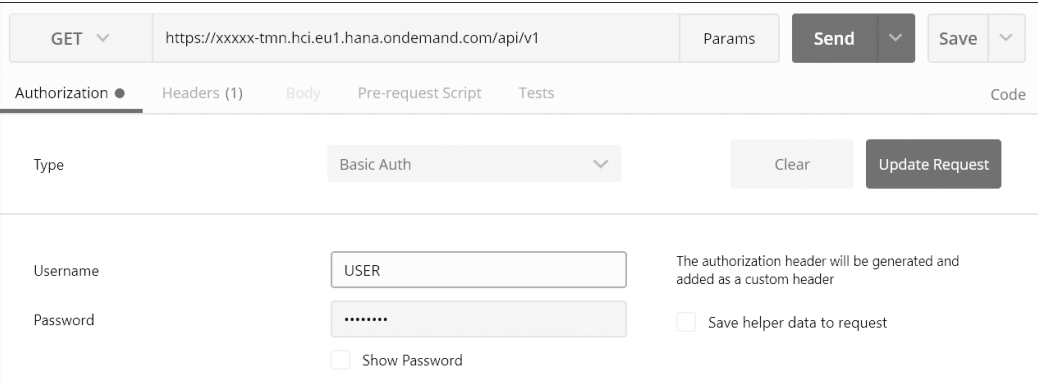


Figure 9.38 Configuring Endpoint and Authorization in Postman for Neo

Under the **Headers** tab, add a new key named **X-CSRF-Token** with the value `Fetch` to request an X-CSRF token, as shown in Figure 9.39.

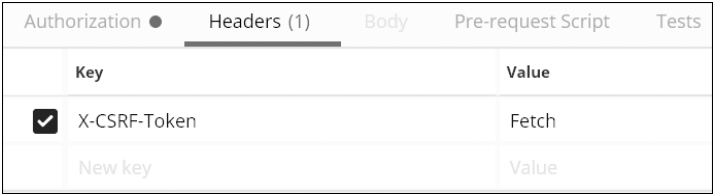


Figure 9.39 Adding the X-CSRF-Token Header

Click the **Send** button (shown earlier in Figure 9.38 in the top right) to trigger the request. The response message includes a number of headers, including the X-CSRF token, which can be identified by the label `X-CSRF-Token` under the **Headers** tab, as shown in Figure 9.40.

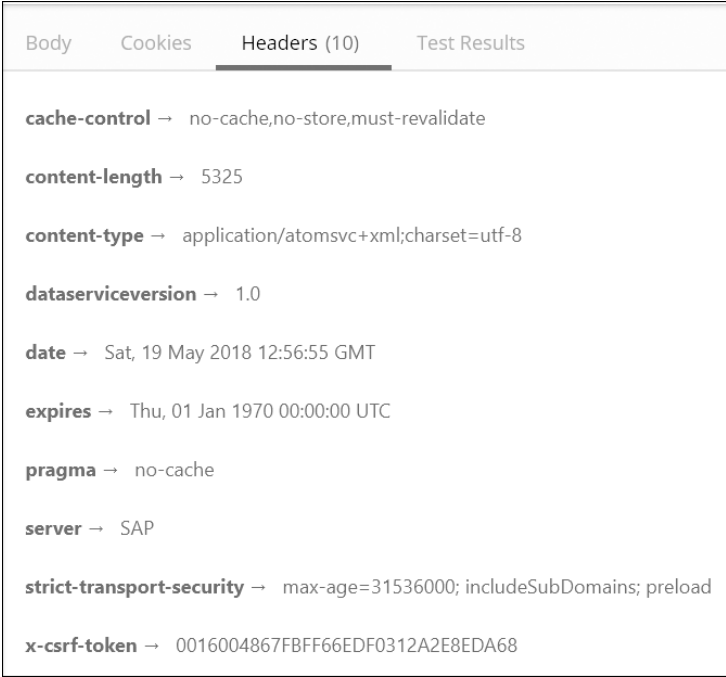


Figure 9.40 X-CSRF-Token in the Headers Tab

You can now use value of the X-CSRF token in the header of your next OData API request. Furthermore, the body of the response message is filled with a list of entity types that the X-CSRF token can be used with. In other words, the scope of the X-CSRF token is limited to the entity types listed in the response, as shown in Figure 9.41.

In the following sections, we’ll explore the following different API categories:

- Monitoring message flows using the API
- Managing deployed integration content using the API
- Managing log files using the API
- Managing the message store using the API
- Managing security material using the API
- Managing the Partner Directory using the API

Let’s start with message flow APIs.

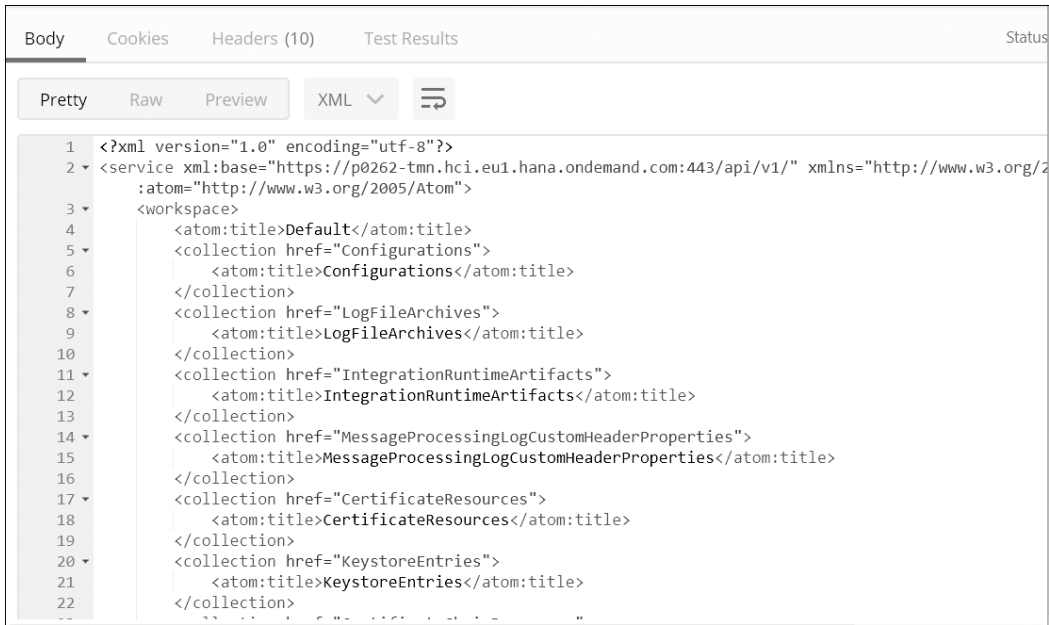


Figure 9.41 Response Body Associated with the X-CSRF Token

9.5.3 Monitoring Message Flows Using the API

In Chapter 8, Section 8.2.3, we described how you can monitor message processing. In APIs related to monitoring message flows, the following entity types play key roles:

- **MessageProcessingLogs**
Entity responsible for MPLs. This entity is the main and parent entity of the API. From this entity, you can navigate to all other entities.
- **MessageProcessingLogAdapterAttributes**
Encapsulates the adapter attributes of the MPL of a specified message entry. This entity includes details such as the type of adapter used in the related integration flow.
- **MessageProcessingLogAttachments**
Contains attachments of the MPL related to a specified message entry.
- **MessageProcessingLogCustomHeaderProperties**
Contains custom header properties of MPLs related to a specified message entry.
- **MessageProcessingLogErrorInformation**
Contains error information for the message related to a specified message entry.

Many APIs are included in these entities. Describing all of these APIs in detail is beyond the scope of this chapter. However, you can find extensive details and examples in the SAP documentation at <http://s-prs.co/507755>. You can also explore these APIs via the SAP API Business Hub at <http://s-prs.co/507756>.

Figure 9.42 shows the entities and APIs available for MPLs. You can test and explore each API using one of the testing approaches we explored in Section 9.5.1.

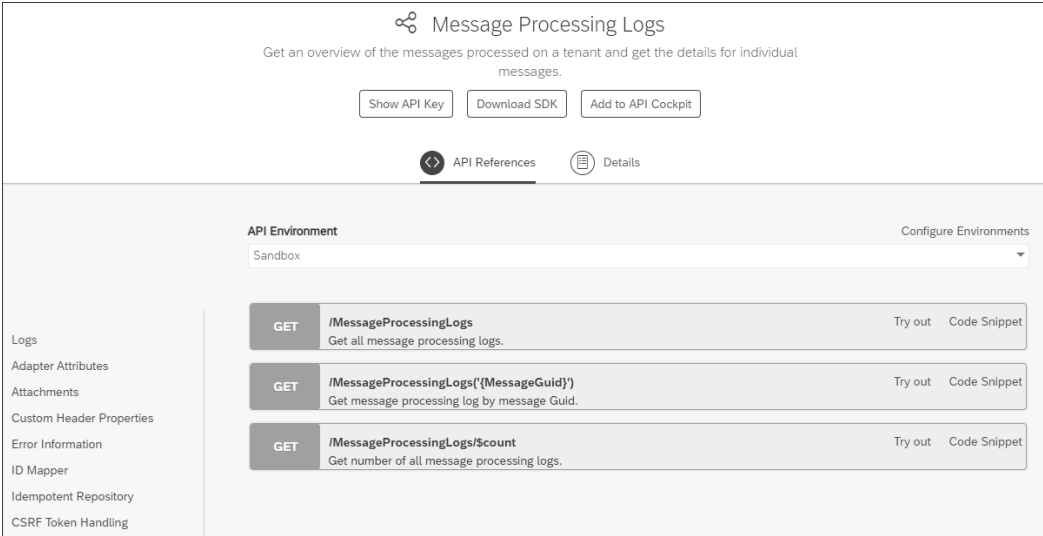


Figure 9.42 Entity Types and APIs in the MPLs

Let's walk through an example scenario to illustrate some functionalities and the usage of APIs related to the monitoring of message flows.

We'll assume that your organization uses many integration platforms, including SAP Process Orchestration, SAP Cloud Platform Integration, and other third-party platforms. You've been asked to build a custom dashboard monitoring solution so that users can see, at a glance, statistics and lists of messages failing in the various integration platforms. The advantage of this dashboard is that the user won't need to log on to each of these platforms individually. Instead, after logging on to the custom dashboard, the user will see errors as they occur on all integration platforms.

How you can programmatically retrieve the relevant information from the MPLs in SAP Cloud Platform Integration and find the entries with errors? Monitoring message flows APIs come to the rescue.

To solve this challenge, you'll need an API that retrieves all MPLs in an error state from the last hour, which will require the use of the MessageProcessingLogs entity. One potential solution is to use the following OData endpoint:

```
https://<tokenurl>/api/v1/MessageProcessingLogs?$inlinecount=allpages&$filter=
Status eq 'FAILED ' and LogStart gt datetime '2020-05-24T12:00:00 ' and LogEnd
lt datetime '2020-05-24T13:00:00 ' &$expand=AdapterAttributes
```

Let’s examine this OData endpoint with the help of the attributes listed in Table 9.12 to understand what’s happening. (Note that some attributes described in Table 9.12 were also previously explained in Table 9.6.)

API Endpoint Element	Description	Example
MessageProcessingLogs	Retrieves MPL entries.	MessageProcessingLogs
inlinecount	Indicates that the response should contain a count of the number of entries in the returned collection.	allpages
filter	Filters the result based on various criteria. In the example column, we’re filtering for all messages that have the status Failed . Additionally, we’re filtering for all message logs that have been created between 29/04/2018 at 12:00:00 and 29/04/2018 at 13:00:00.	Status eq ‘FAILED’ and Log-Start gt datetime’2018-04-29T12:00:00’ and LogEnd lt datetime’2018-04-29T13:00:00’
expand	Retrieves correlated entities for a given navigation. In our case, we also want to retrieve adapter-specific attributes.	AdapterAttributes

Table 9.12 Attributes Included in the OData Endpoint to Retrieve Entries with Errors

After calling the OData endpoint that solves our challenge, you’ll get the response message shown in Listing 9.2.

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xml:base="https://p0262-tmn.hci.eu1.hana.ondemand.com:443/api/v1/">
<id>
https://p0262-tmn.hci.eu1.hana.ondemand.com:443/api/v1/MessageProcessingLogs
</id>
<title type="text">MessageProcessingLogs</title>
<updated>2018-04-29T12:58:12.413Z</updated>
<author>
<name/>
</author>
<link href="MessageProcessingLogs" rel="self" title="MessageProcessingLogs"/>
```

```
<m:count>1</m:count>
<entry>
<id>
https://p0262-tmn.hci.eu1.hana.ondemand.com:443/api/v1/
MessageProcessingLogs('Afr1v7P0JUYGSI2bXJAGRQ74HMyP')
</id>
<title type="text">MessageProcessingLogs</title>
<updated>2018-04-29T12:58:12.413Z</updated>
<category term="com.sap.hci.api.MessageProcessingLog" scheme="http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<link href="MessageProcessingLogs('Afr1v7P0JUYGSI2bXJAGRQ74HMyP')" rel="edit"
title="MessageProcessingLog"/>
<link href="MessageProcessingLogs('Afr1v7P0JUYGSI2bXJAGRQ74tiHyP')/
CustomHeaderProperties" rel="http://schemas.microsoft.com/ado/2007/08/
dataservices/related/CustomHeaderProperties" title="CustomHeaderProperties"
type="application/atom+xml;type=feed"/>
<link href="MessageProcessingLogs('Afr1v7P0JUYGSI2bXJAGRQ74HMyP')/
MessageStoreEntries" rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
related/MessageStoreEntries" title="MessageStoreEntries" type="application/
atom+xml;type=feed"/>
<link href="MessageProcessingLogs('Afr1v7P0JUYGSI2bXJAGRQ74tiHyP')/
ErrorInformation" rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
related/ErrorInformation" title="Errorinformation" type="application/
atom+xml;type=entry"/>
<link href="MessageProcessingLogs('Afr1v7P0JUYGSI2bXJAGRQ74HMyP')/
AdapterAttributes" rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
related/AdapterAttributes" title="AdapterAttributes" type="application/
atom+xml;type=feed">
<m:inline>...</m:inline>
</link>
<link href="MessageProcessingLogs('Afr1v7P0JUYGSI2bXJAGRQ74tiHyP')/Attachments"
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Attachments"
title="Attachments" type="application/atom+xml;type=feed"/>
<link href="MessageProcessingLogs('Afr1v7P0JUYGSI2bXJAGRQ74HMyP')/Runs"
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Runs"
title="Runs" type="application/atom+xml;type=feed"/>
<content type="application/xml">
<m:properties>
<d:MessageGuid>Afr1v7P0JUYGSI2bXJAGRQ74HMyP</d:MessageGuid>
<d:CorrelationId>Afr1v7MGmzLrpFr0xq00lvAcig5Y</d:CorrelationId>
<d:ApplicationMessageId m:null="true"/>
<d:ApplicationMessageType m:null="true"/>
<d:LogStart>2018-04-29T12:50:59.723</d:LogStart>
<d:LogEnd>2018-04-29T12:51:00.12</d:LogEnd>
```

```
<d:Sender>Sender_SOAP</d:Sender>
<d:Receiver m:null="true"/>
  <d:IntegrationFlowName>Invoking_OData</d:IntegrationFlowName>
<d:Status>FAILED</d:Status>
<d:AlternateWebLink>...</d:AlternateWebLink>
<d:IntegrationArtifact m:type="com.sap.hci.api.IntegrationArtifact">
  <d:Id>Invoking_OData</d:Id>
  <d:Name>Invoking_OData</d:Name>
  <d:Type>INTEGRATION_FLOW</d:Type>
</d:IntegrationArtifact>
<d:LogLevel>INFO</d:LogLevel>
<d:CustomStatus>FAILED</d:CustomStatus>
</m:properties>
</content>
</entry>
</feed>
```

Listing 9.2 Response of the OData Endpoint Call

When examining the response message shown in Listing 9.2, you’ll notice an element named **entry**, which represents a log entry in the message monitor. Note that, if multiple entries are returned, the entries are sorted in descending order (with the oldest entry on the top) by default. The entries returned in this response can also be found in the SAP Cloud Platform Integration Web UI, on the **Monitor** page.

In addition, notice that the entry shown in Listing 9.2 has a **MessageGuid** with a value **AFr1v7POJUYGSI2bXJAGRQ74HMyP**. The same entry can also be found in the **Monitor Message Processing** section of SAP Cloud Platform Integration, shown in Figure 9.43, next to the **Message ID** field. Also note that the response message has a field labeled **count**, which specifies the number of returned entries. Furthermore, note that the field **IntegrationFlowName** has the value **Invoking_OData**, which will be of use later in Section 9.5.4.

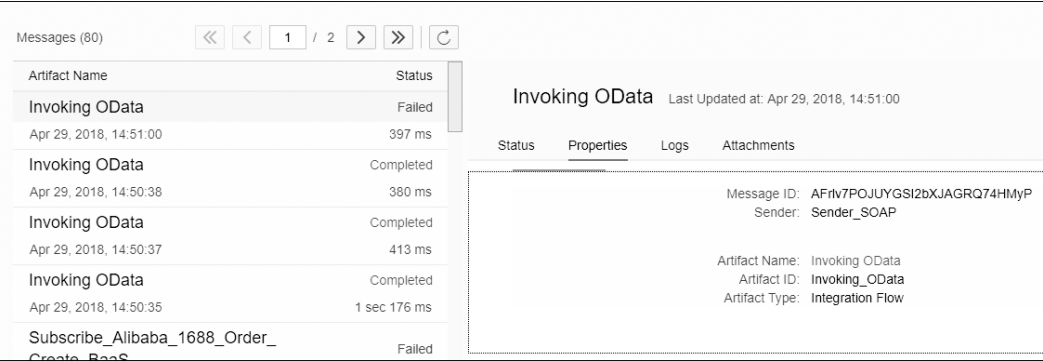


Figure 9.43 MPLs in SAP Cloud Platform Integration

The entry returned in the OData API call contains a number of properties, as shown earlier in Listing 9.2. Table 9.13 lists the **MessageProcessingLogs** properties and their descriptions.

Property	Description
MessageGuid	GUID of the message that the processing log concerns.
CorrelationId	GUID of the correlated messages.
ApplicationMessageId	GUID specific to a particular application. Think about this value as an identifier set for the sake of identification by an external application. This value can be set using a Content Modifier step and assigning a value to the SAP_ApplicationID header element.
ApplicationMessageType	Property to represent a type of message as known by a business application. Use a Script step in the integration flow to set this property.
LogStart	Date and time that the writing of the log started.
LogEnd	Date and time that the writing of the log ended.
Sender	Identifier of the sender system.
Receiver	Identifier of the receiver system.
IntegrationFlowName	Name of the integration flow.
Status	Status of the message processing. Currently, the following statuses are possible: COMPLETED , PROCESSING , RETRY , ERROR , ESCALATED , and FAILED .
AlternateWebLink	Link used to directly open the MPL on this monitoring entry.
IntegrationArtifact/Id	Technical name or ID of the integration flow.
IntegrationArtifact/Name	Name of the integration flow. This value is identical to the IntegrationFlowName property.
IntegrationArtifact/Type	Type of artifact that this message processing concerns, for example, INTEGRATION_FLOW .

Table 9.13 Properties of the MessageProcessingLogs

In the following sections, we’ll continue with our OData API journey by exploring APIs that relate to deployed integration content.

9.5.4 Managing Deployed Integration Content Using the API

Using the OData APIs provided by SAP Cloud Platform Integration, you can query the content of integration artifacts deployed on a tenant. The APIs that access the deployed integration content revolve around the following entity types:

- **IntegrationRuntimeArtifact**
Manages all deployed integration artifacts in the tenant. It’s also possible to use the POST method to deploy an artifact from the file system. Additionally, an already deployed artifact can be undeployed using the DELETE method.
- **IntegrationRuntimeArtifactsErrorInformation**
Holds error information of a specific deployed integration artifact.
- **CSRF Token Handling**
Holds the X-CSRF token for this session. The X-CSRF token is only required for write access (as discussed in Section 9.5.2).

Too many APIs use these entities to discuss them all in this chapter. However, you can find extensive details and examples in the SAP documentation at <http://s-prs.co/507757>. You can also explore these APIs via the SAP API Business Hub at <http://s-prs.co/507758>.

Figure 9.44 shows the entities and APIs available. You can test and explore each API using one of the testing approaches we explored in Section 9.5.1.

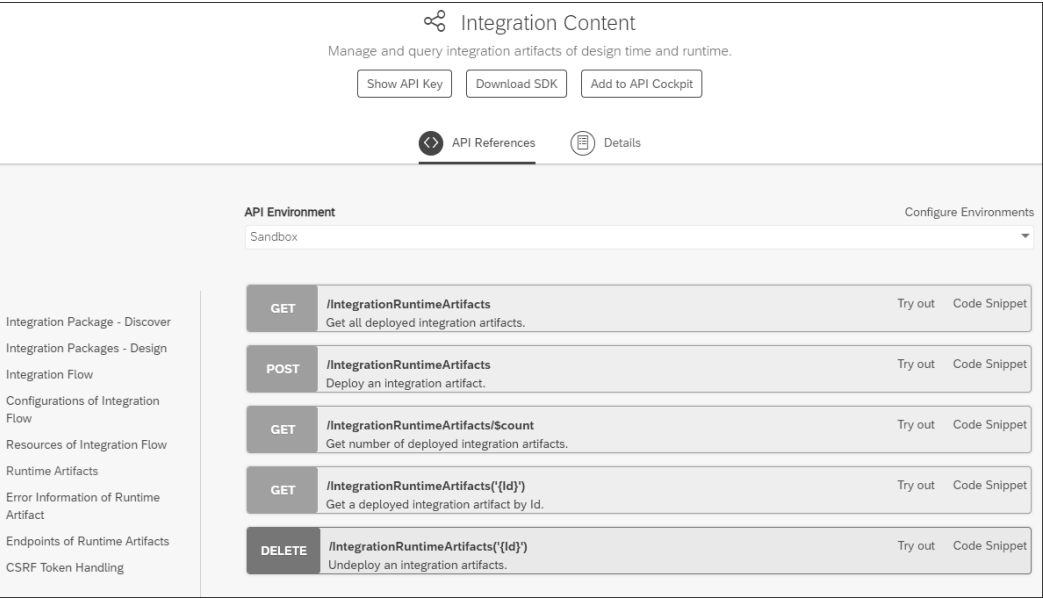


Figure 9.44 Entity Types and APIs Related to Deployed Integration Content

Let’s further enhance the example scenario from Section 9.5.3 to illustrate the usage and functionalities for managing deployed integration content. In the previous

section, we retrieved an entry with an error using the APIs for MPLs. Imagine that, after retrieving and displaying an entry with an error in your custom dashboard, you now also want to see details about the related deployed integration content. Perhaps, you’re interested to know the name, status, and version of the deployed content, as well as the user who deployed it and when.

To solve this challenge, you’ll need to use the IntegrationRuntimeArtifact entity. One solution is to use the following OData endpoint:

`https://< tokenurl>/api/v1/IntegrationRuntimeArtifacts('Invoking_OData').`

Note the value Invoking_OData was retrieved from the field IntegrationFlowName within the content node, as shown in Listing 9.2.

The resulting response of the preceding OData endpoint is shown in Figure 9.45.

```
<?xml version='1.0' encoding='utf-8'>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xml:base="https://p0262-tmn.hci.eu1.hana.ondemand.com:443/api/v1/">
  <id>
    https://p0262-tmn.hci.eu1.hana.ondemand.com:443/api/v1/IntegrationRuntimeArtifacts('Invoking_OData')
  </id>
  <title type="text">IntegrationRuntimeArtifacts</title>
  <updated>2018-04-29T15:49:08.406Z</updated>
  <category term="com.sap.hci.api.IntegrationRuntimeArtifact" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  </category>
  <link href="IntegrationRuntimeArtifacts('Invoking_OData')" rel="edit" title="IntegrationRuntimeArtifact"/>
  <link href="IntegrationRuntimeArtifacts('Invoking_OData')/$value" rel="edit-media" type="application/octet-stream"/>
  <link href="IntegrationRuntimeArtifacts('Invoking_OData')/ErrorInformation"
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/ErrorInformation"
title="ErrorInformation"
type="application/atom+xml;type=entry"/>
  <content type="application/octet-stream" src="IntegrationRuntimeArtifacts('Invoking_OData')/$value"/>
  <m:properties>
    <d:Id>Invoking_OData</d:Id>
    <d:Version>1.0.0</d:Version>
    <d:Name>Invoking_OData</d:Name>
    <d:Type>INTEGRATION_FLOW</d:Type>
    <d:DeployedBy>S0011540061</d:DeployedBy>
    <d:DeployedOn>2018-04-28T21:11:25.951</d:DeployedOn>
    <d:Status>STARTED</d:Status>
  </m:properties>
</entry>
```

Figure 9.45 Response of the OData Endpoint Call

Note that every <entry> element returned in the response shown in Figure 9.45, represents an artifact in SAP Cloud Platform Integration. In our case, we only have one entry returned. The properties element of the response message shown in Figure 9.45 includes a number of attributes to describe the deployed integration content. These properties are listed in Table 9.14.

Attributes	Description
Id	Technical identification of the integration content.
Version	Latest version of the integration content when deployed.
Name	Name of the integration
Type	Type of artifact that this message processing concerns. Possible values include INTEGRATION_FLOW, VALUE_MAPPING, DATA_INTEGRATION, and ODATA_SERVICE.

Table 9.14 Properties Available for the Deployed Integration Content OData API

Attributes	Description
DeployedBy	Name of the user who deployed the content.
DeployedOn	Date and time that the integration content was last deployed.
Status	Current status of deployed integration content. Possible values include STARTED, STARTING, and ERROR.

Table 9.14 Properties Available for the Deployed Integration Content OData API (Cont.)

In the next section, we’ll explore APIs that relate to log files.

9.5.5 Managing Log Files Using the APIs

Using the OData APIs provided by SAP Cloud Platform Integration, you can query log files on a tenant. Note that, at the time of this writing, the Log Files API is only available in the Neo environment and not yet in Cloud Foundry.

Note that log files come in two types:

- **Default trace**
These log files include processing information of a technical nature.
- **HTTP access logs**
These log files include information about all inbound HTTP requests arriving in SAP Cloud Platform Integration.

The APIs that facilitate the access to log files revolve around two main entity types:

- **LogFileArchives**
Used for all archived log files.
- **LogFiles**
Used for all current (nonarchived) log files.

These entities include a number of APIs. We won’t explore them all in this section, but we’ll look at a scenario to showcase what’s possible. To get a full description of the different APIs, refer to the SAP documentation at <http://s-prs.co/507759>. You can also explore these APIs via the SAP API Business Hub at <http://s-prs.co/507760>.

Figure 9.46 shows the available entities and APIs. You can test and explore each API using one of the testing approaches we explored in Section 9.5.1.

Let’s return to our example scenario to illustrate the usage and functionalities of APIs relating to log files. In Section 9.5.3, we retrieved an entry with an error using the MPL API. Imagine that you want to further troubleshoot the error from your custom dashboard. For this task, you want to download a copy of all log files of type HTTP around the time that the error occurred. Note that the error occurred around 12:51, as indicated by the `LogStart` field shown earlier in Listing 9.2.

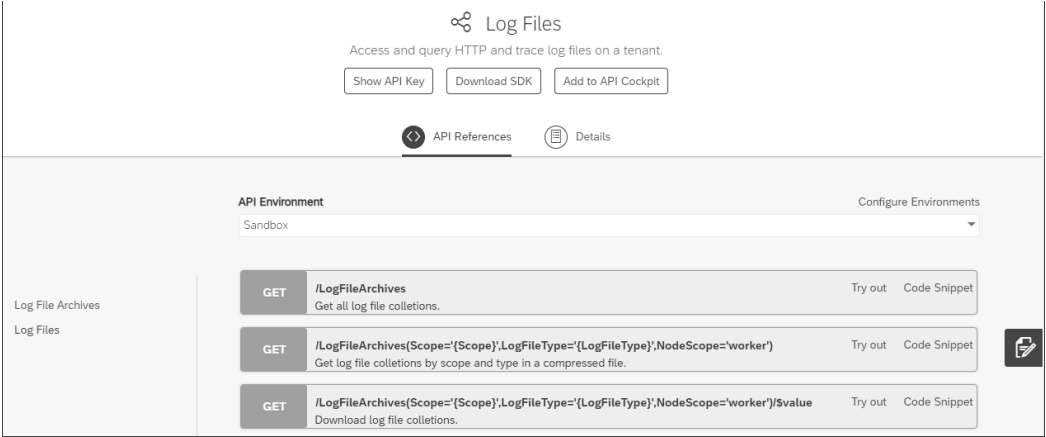


Figure 9.46 Entity Types and APIs Related to Log Files

To solve this challenge, we can use the `LogFileArchives` entity by invoking the following OData endpoint:

`https://<tokenurl>/api/v1/LogFileArchives(Scope='all',LogFileType='http',NodeScope='worker')/$value?modifiedAfter=2018-04-29T12:50:00Z`

Let’s now examine this OData endpoint in detail to understand what is happening, referring to the details listed in Table 9.15.

Endpoint Attribute	Description
LogFileArchives	Entity used to retrieve an archived log file.
Scope	Indicates which scope/type of log files you want to download. Possible values include <code>all</code> (to download all existing HTTP log files) and <code>latest</code> (to only retrieve the latest HTTP log files).
LogFileType	Filters the result based on the type of log file. Possible values include <code>http</code> and <code>Trace</code> .
NodeScope	Specifies that we’re only interested in retrieving log files from runtime nodes (also referred to as worker nodes).
value	Specifies that the next parameter in the URL will contain parameter values.
modifiedAfter	Specifies the time after which the filtered log file was changed.

Table 9.15 Attributes Included in the OData Endpoint to Log Archive Entries

Note that you must have the role `IntegrationOperationServer.read` assigned to your user to call log file APIs.

In the next section, we’ll explore APIs that relate to the message store.

9.5.6 Managing Message Store Entries Using APIs

Using the OData APIs provided by SAP Cloud Platform Integration, you can access the tenant’s message store entries. In scenarios with the requirement to persist messages, message content can be written and saved in the message store using the **Persist Message** step of an integration flow. You can then access the stored message and analyze it later. However, note that a message is stored on the runtime for a maximum of 90 days. After this time, the message is automatically deleted.

Note
At the time of this writing, no user interface (UI) available for the message store. The OData API is the only option for accessing the content of a message store.

For each entry in a message store, you can retrieve its properties, headers, payload, and attachments. The APIs that access message stores revolve around the following four main entity types:

- **MessageStoreEntries**
Used to manage message store entries.
- **MessageStoreEntryProperties**
Used to manage properties of message store entries.
- **MessageStoreEntryAttachments**
Used to manage attachments from a specific message store entry.
- **MessageStoreEntryAttachmentProperties**
Used to manage properties of an attachment in the message store.

Referring to the entity model diagram shown earlier in Figure 9.19, notice that a direct relationship exists between a message store entry (represented by the entity type `MessageStoreEntries`) and `MessageProcessingLogs`. This relationship means that, for every entry in the processing log with an attachment, you can try to retrieve its related message store entries (if available).

In this section, we won’t explore all APIs involving in the entities listed earlier, but we’ll look at a scenario to showcase what is possible. To get a full description of the different APIs, refer to the SAP documentation at <http://s-prs.co/507761>. You can also explore these APIs via the SAP API Business Hub at <http://s-prs.co/507762>.

Figure 9.47 shows the entities and APIs available. You can test and explore each one of these APIs using the testing approaches we explored in Section 9.5.1.

Note that you must have the role `esbmessagestorage.read` assigned to your user to call message store APIs.

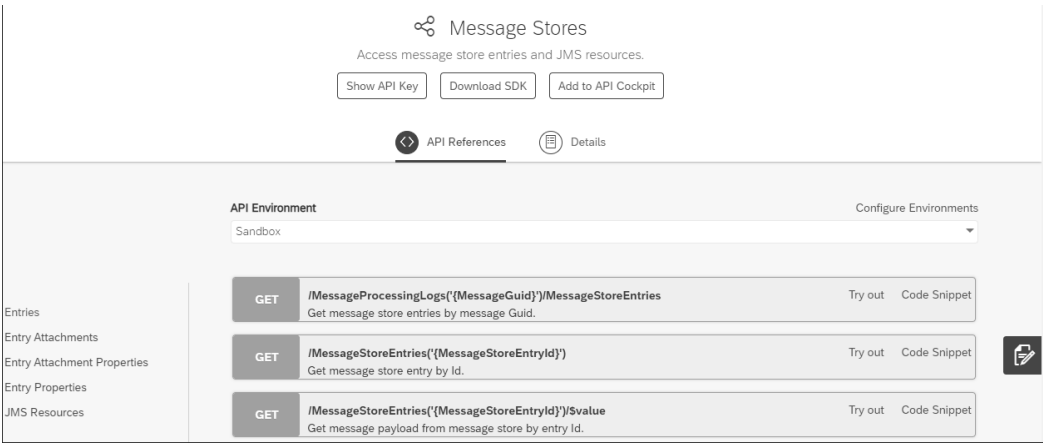


Figure 9.47 Entity Types and APIs Related to the Message Store

Consider the message aggregation scenario we explored in Chapter 4, Section 4.6. In that scenario, we aggregated correlated messages. Imagine that, after the aggregation process is finished, we want to persist the final aggregated payload in the message store. We can write the payload to the message store using the **Persist Message** step to our integration flow.

To start developing this integration flow, follow these steps:

1. On the **Design** page of SAP Cloud Platform Integration, open the integration flow that we created in Chapter 4, Section 4.6.
2. Switch to the edit mode by clicking the **Edit** button in the top-right corner of the integration flow screen.
3. Add the **Persist Message** step to the integration flow. The **Persist** step can be found in the palette on the left, as shown in Figure 9.48.

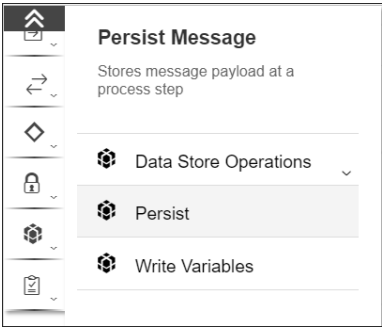


Figure 9.48 Selecting the Persist Step from the Palette

4. Ensure that the **Step ID** of the newly added step is unique. The final integration flow should be similar to the processes shown in Figure 9.49 and Figure 9.50. Note that,

according to our integration flow, the message store is only populated after all messages have been collected because the **Persist** step comes after the **Aggregator** step.

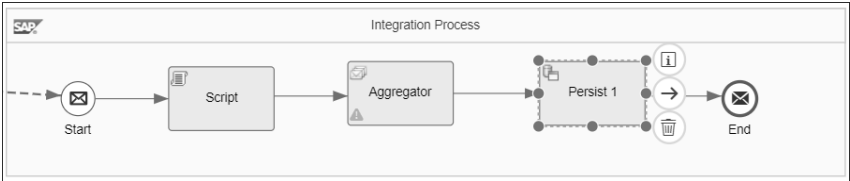


Figure 9.49 Overview of the Integration Flow Extended with a Persist Step A

Persist Message	
General	Processing
Step ID:* Persist1	
Encrypt Stored Message:	<input checked="" type="checkbox"/>

Figure 9.50 Overview of the Integration Flow Extended with a Persist Step B

5. Save and deploy the integration flow.

Now, let's assume that, in our custom dashboard application that we started building in Section 9.5.3, we want to be warned if a failed message flow contains entries in the message store. In this case, you'll retrieve the payload of the entry in the message store.

To retrieve the payload of the entry in the message store, you'll need to call several OData APIs in the following sequence:

1. Use `MessageProcessingLogs` to retrieve the list of failing messages. You already know how to query for failed messages from our discussion in Section 9.5.3.
2. Use the message `Guid` of the message retrieved from the first call to make a second call to query if there are message store entries for messages with the specified message `Guid`. In this step, you can use the following endpoint:

`https://<tenant>/api/v1/MessageProcessingLogs('<Guid>')/MessageStoreEntries`

The response of the API call is shown in Listing 9.3.

```
<content type="application/octet-stream" src="MessageStoreEntries("sap-it-res%3Amsg%3Aac965bd8f%3Abe694f69-73a5-4430-b681-1673c963fd4c")/$value"/>
```

Listing 9.3 Response of the OData API Call

3. Retrieve the payload of the entry in the message store. Looking at the entry returned in Listing 9.3, notice the `src` attribute of the `content` element. The value of this attribute provides details regarding how to retrieve the payload. In this example, use the following link to retrieve the payload:

`https://<tokenurl>/api/v1/MessageStoreEntries('sap-it-res%3Amsg%3Aac965bd8f%3Abe694f69-73a5-4430-b681-1673c963fd4c')/$value`

Note that, in this URL, the value between `/v1/` and `/$value` is copied from the `src` attribute of Listing 9.3. The API returns the aggregated payload as shown in Figure 9.51.

```
<?xml version="1.0" encoding="UTF-8"?><multimap:Messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge"><multimap:Message1><OrderItem xmlns:demo="http://hci.sap.com/demo" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><orderNumber>AA2345</orderNumber><Item><ItemNo>1</ItemNo><Quantity>1</Quantity><Unit>1</Unit><LastStatus>false</LastStatus></Item></OrderItem><OrderItem xmlns:demo="http://hci.sap.com/demo" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><orderNumber>AA2345</orderNumber><Item><ItemNo>2</ItemNo><Quantity>5</Quantity><Unit>1</Unit><LastStatus>false</LastStatus></Item></OrderItem><OrderItem xmlns:demo="http://hci.sap.com/demo" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><orderNumber>AA2345</orderNumber><Item><ItemNo>3</ItemNo><Quantity>25</Quantity><Unit>1</Unit><LastStatus>true</LastStatus></Item></OrderItem></multimap:Message1></multimap:Messages>
```

Figure 9.51 Aggregated Payload Returned by the Message Store API

Now that you know how to use the message store API, let's explore the OData APIs related to security materials.

9.5.7 Managing Security Material Using the API

Using the OData APIs provided by SAP Cloud Platform Integration, we can access key-store content and other security-related artifacts, for example, **User Credentials** artifacts. This API contains a lot of features that can't all be explored in this section.

For a full description of the different APIs available for managing security material, refer to the SAP documentation via <http://s-prs.co/507763>.

To give you an idea of how to use these APIs, let's work through an example scenario to illustrate its usage. Let's assume that you want your keystore entries to be automatically backed up at the end of each month. Given that you don't want to perform this activity manually every month, you're looking for some way to automate the process via your custom dashboard application. (Note that we discussed the topic of keystores in detail in Chapter 8, Section 8.3.2.)

You can easily achieve this automation task by getting your custom application to call the Security Material API. More specifically, you can use an API that enables backs up all keystore entries via the endpoint:

`https://<tokenurl>/api/v1/KeystoreResources`

Note that you’ll need to use a POST method for this request. Listing 9.4 shows an example request. You can also include, in the request, the query option indicated in Table 9.16.

```
{ "Name": "backup_admin_system" }
```

Listing 9.4 Example Request Body to Back Up Keystore Entries

Query Option	Description
returnKeystoreEntries	Possible values include true and false. When set to true, the KeystoreEntry instances that have been backed up are returned in the response. Note that this query is optional and defaults to false.

Table 9.16 Possible Query Option for Renaming an Alias

Because the keystore OData API is protected against CSRF attacks, you must first fetch an X-CSRF token before you can make this API call. (We explored how to fetch X-CSRF tokens in Chapter 7, Section 7.4.3.)

Let’s now explore APIs that relate to managing the Partner Directory in the next section.

9.5.8 Managing the Partner Directory Using the API

In Chapter 7, Section 7.4, we introduced you to the tenant Partner Directory. We described how, during a business-to-business (B2B) project, an SAP Cloud Platform Integration owner might decide to build an application where partners involved in the scenario can maintain their own configuration data.

At the time of this writing, Partner Directory information can only be maintained via an OData API. The HTTP addresses required to make outbound calls to the partner systems are examples of the type of data stored in the Partner Directory.

Assuming that the purpose of the Partner Directory is clear to you, we’ll now focus on using the Partner Directory OData APIs provided by SAP Cloud Platform Integration. These APIs access the Partner Directory, create entries, and help manage them. These APIs revolve around the following entity types:

- AlternativePartners
- AuthorizedUsers
- BinaryParameters
- Partners
- StringParameters
- UserCredentialParameters
- CSRF Token Handling

Updated details about these APIs can be found via the SAP API Business Hub at <https://api.sap.com/api/PartnerDirectory>.

Figure 9.52 shows the entities and APIs available. You can test and explore each of these APIs using one of the testing approaches we explored in Section 9.5.1.

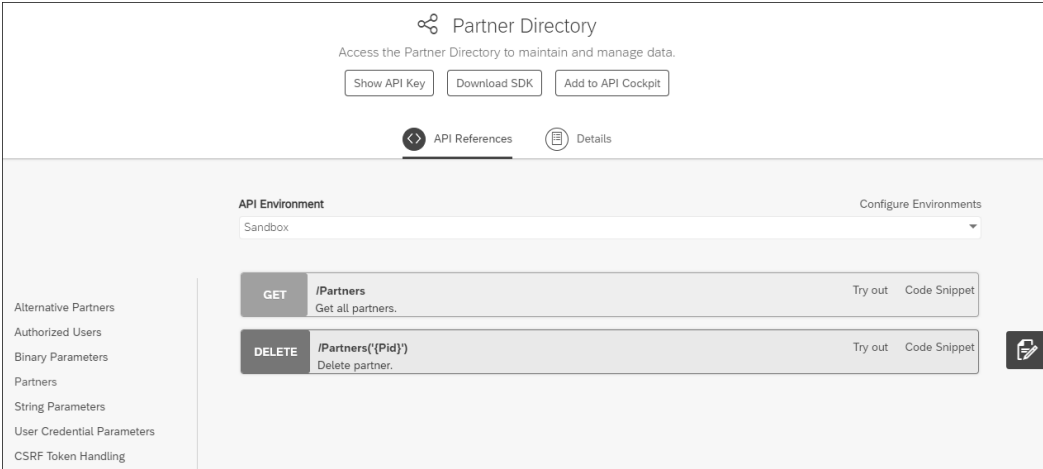


Figure 9.52 Entity Types and APIs Related to the Partner Directory

To get an idea of how to use these APIs, refer to the sample scenario in Chapter 7, Section 7.4. That scenario used the OData API to store an endpoint URL and a credential alias in Partner Directory. We therefore won’t repeat these details in this section.

Note

Be aware of the following Partner Directory limitations:

- The number of AlternativePartners in the tenant is limited to a maximum of 1,000,000.
- The number of AuthorizedUsers in the tenant is limited to a maximum of 500,000.
- The number of BinaryParameters in the tenant is limited to a maximum of 400,000.
- The number of StringParameters in the tenant is limited to a maximum of 3,000,000.

9.6 Using SAP Cloud Platform Integration with SAP Cloud Platform API Management

In the past, APIs were mostly only known to programmers, but in today’s digital era, even business executives are aware of APIs and their potential financial impacts. In a digitized world, many companies are generating revenue by exposing their APIs to business partners, suppliers, and customers like they would for any other service offering.

Companies like Amazon, Facebook, Twitter, Netflix, Uber, and Google are generating huge revenues based on their APIs. So, chances are high that APIs will play a key role in the digital transformation journey of your organization. Today, APIs are managed like traditional products!

SAP Cloud Platform API Management (SAP API Management) is, next to SAP Cloud Platform Integration, a key constituent of the SAP Cloud Platform Integration Suite. The SAP API Management solution can help you in your digital transformation journey by providing simple, scalable, and secure access to your organization's digital assets through APIs. SAP API Management enables developer communities to consume and discover your organization's APIs. Refer to <http://s-prs.co/5077119> to read more about SAP Cloud Platform API Management.

Some key capabilities of SAP API Management, include the following features, just to name a few:

- The ability to provision APIs via REST, OData, and SOAP in a standardized and consistent way
- Real-time and historic analytics on API usage, errors, monitoring, and traffic
- High security standards for the APIs to prevent against attacks such as denial-of-service (DoS) attacks, cross-site scripting (XSS), cross-site request forgery (CSRF), and so on
- Robust traffic management of APIs
- Full API lifecycle management
- Management, discovery, testing, subscription, and consumption of APIs by the developer community
- Monetization of APIs

Figure 9.53 shows the positioning of SAP API Management within your landscape. Different applications can consume APIs via SAP API Management, which is acting as a gateway. SAP API Management also proxies the calls to the backend systems (which are either on-premise or cloud-based systems). SAP API Management connects to these backend systems via various protocols, such as SOAP, REST, OData, and so on.

Figure 9.53 shows the different personas involved with SAP API Management, such as the following:

- **External applications (mobile, web, etc.)**
These applications consume the APIs provided by SAP API Management.
- **App developer**
This developer is responsible for making external applications that consume APIs. This developer must be able to discover existing APIs and easily figure out how to consume them.

- **API developer**
This person is responsible for designing and implementing APIs via SAP API Management.
- **API admins and owners**
These people are responsible for administering and managing APIs via monitoring, analyzing, and monetizing processes.

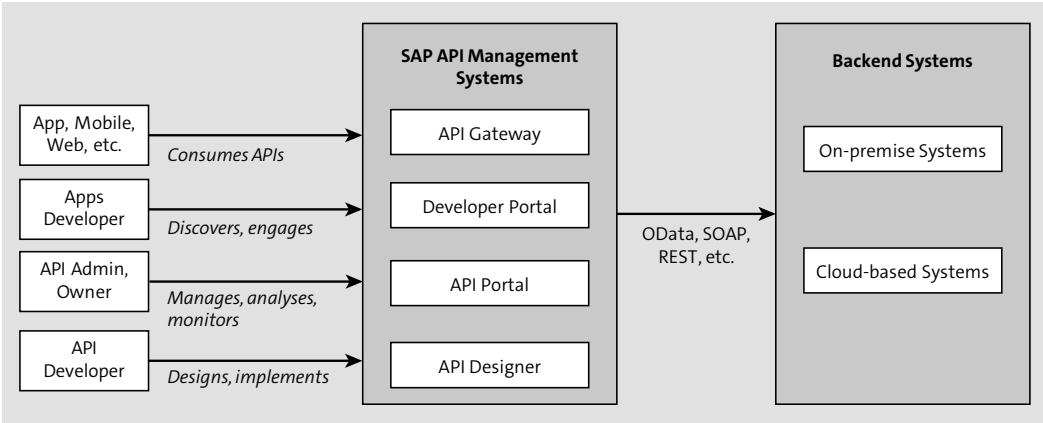


Figure 9.53 Positioning of SAP API Management and Its Personas

The positioning of SAP API Management, as shown in Figure 9.53, also means that SAP API Management can proxy services provided by SAP Cloud Platform Integration, which is the main subject of this section.

SAP API Management is a big topic that deserves its own book. In the following sections, we'll briefly explore how you can use SAP API Management to publish APIs from services provided by SAP Cloud Platform Integration in a secure manner. To find out more about SAP API Management, follow the tutorials available at the SAP Community (www.sap.com/community.html).

Note that SAP API Management also sits on top of SAP Cloud Platform and is included in the SAP Cloud Platform Integration, Enterprise Edition, as discussed in Chapter 1, Section 1.4. You can also register for a free trial account at <https://account.hana-trial.ondemand.com/#/home/welcome>.

You might be wondering how SAP API Management and the SAP API Business Hub are different. SAP API Management enables any organization to expose its own APIs. This solution also allows their business partners to discover and consume these APIs in a secure manner. In contrast, SAP API Business Hub allows you to discover, explore, and test the APIs offered by SAP.

After obtaining your SAP API Management tenant, one of your first tasks will be to establish a connection to your SAP Cloud Platform Integration tenant. Let's explore how this task can be achieved next.

9.6.1 Establishing a Connection between SAP Cloud Platform Integration and SAP API Management

Note that connecting SAP API Management to SAP Cloud Platform Integration is a one-time action. After that, this connection can be reused.

To connect SAP API Management to SAP Cloud Platform Integration, follow these steps:

1. Log on to your SAP API Management tenant via the following URL (if you have a trial account):
<https://account.hanatrial.ondemand.com/cockpit>.
The link to the productive account is given in the tenant provisioning email you received from SAP when getting a new tenant.
If you don't have a trial account, refer to this blog post:
<https://developers.sap.com/tutorials/hcp-apim-enable-service.html>
2. Navigate to the **Services** section from the menu on the left, as shown in Figure 9.54.
3. Click on the **API Management** link under the **Integration** section, as shown in Figure 9.54, in the right panel.

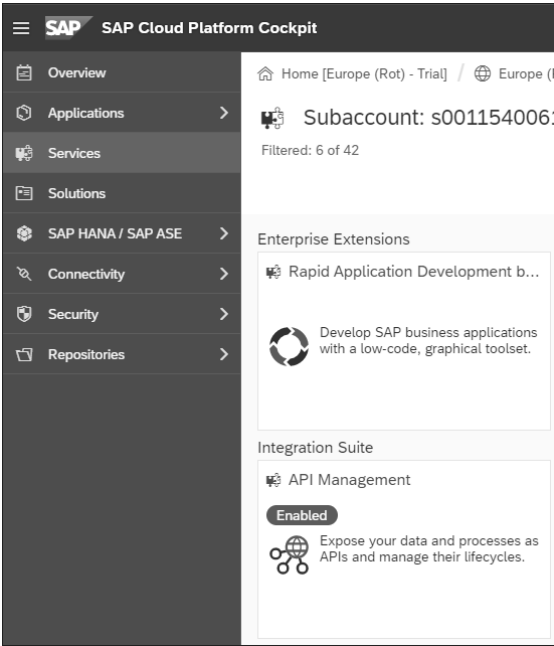


Figure 9.54 Navigating to the SAP API Management Service

4. On the next screen, select the **Access API Portal** link, as shown in Figure 9.55.

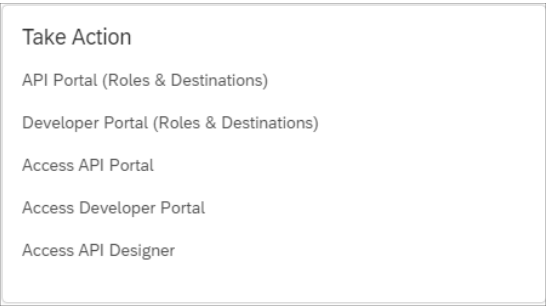


Figure 9.55 SAP API Management Landing Page with an Overview of Possible Actions

5. The next screen presents an overview of the **API Portal** page, which contains information like API traffic, error, usage, performance, applications, and so on. From this page, select **Configure** from the menu on the left, as shown in Figure 9.56.

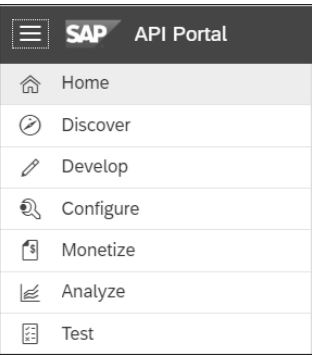


Figure 9.56 Overview of the API Portal

6. On the next page, select the **API Providers** tab, as shown at the top of Figure 9.57. Click the **Create** button to add a new API provider. An API provider represents the backend system that will receive and execute the call.

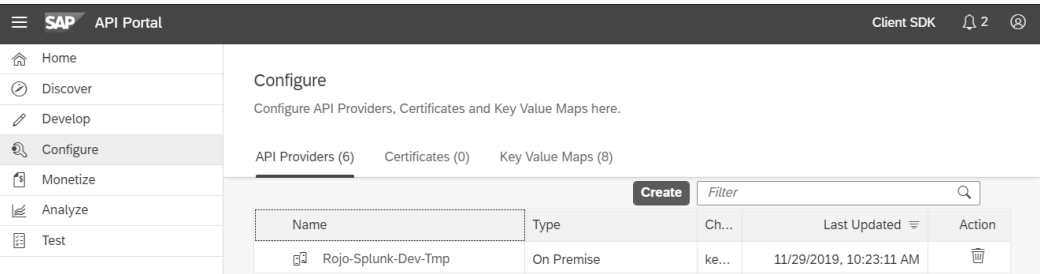


Figure 9.57 API Providers Tab

7. On the next page, provide a name and description for the new API provider, as shown in Figure 9.58.

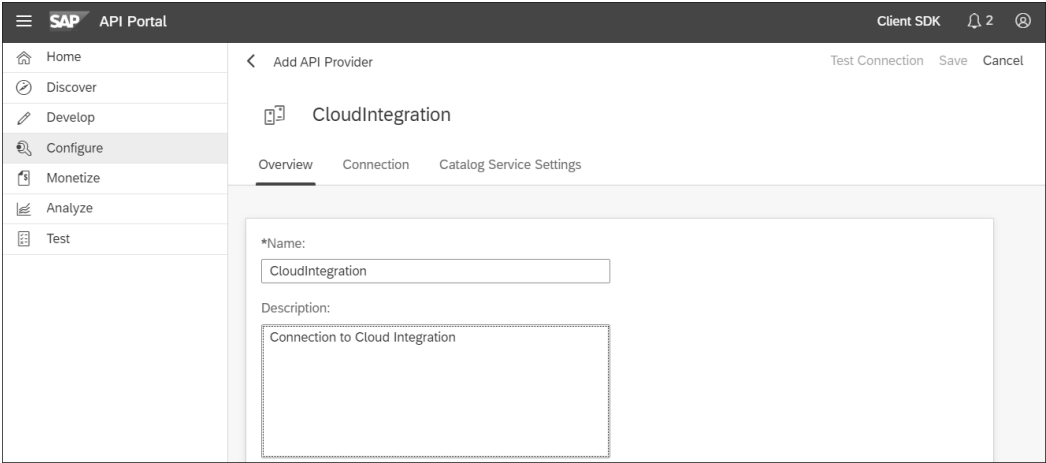


Figure 9.58 Providing a Name and Description

8. Maintain the host name, authentication, and other details related to your SAP Cloud Platform Integration tenant, as shown in Figure 9.59 and Figure 9.60.

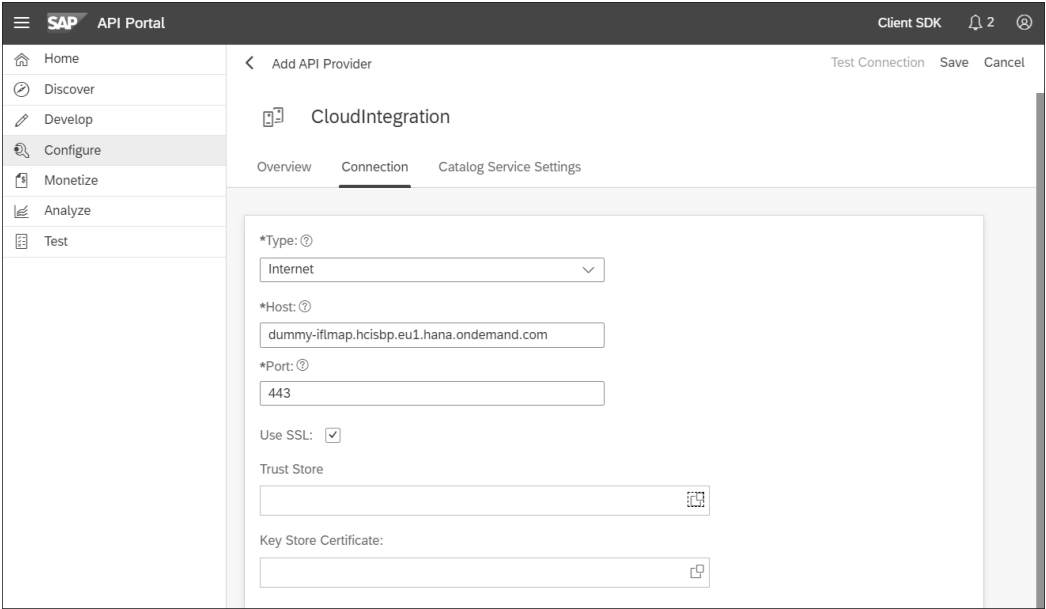


Figure 9.59 Connection Details to the SAP Cloud Platform Integration Tenant

9. Finally, click the **Save** button, as shown in Figure 9.60 in the top-right corner.

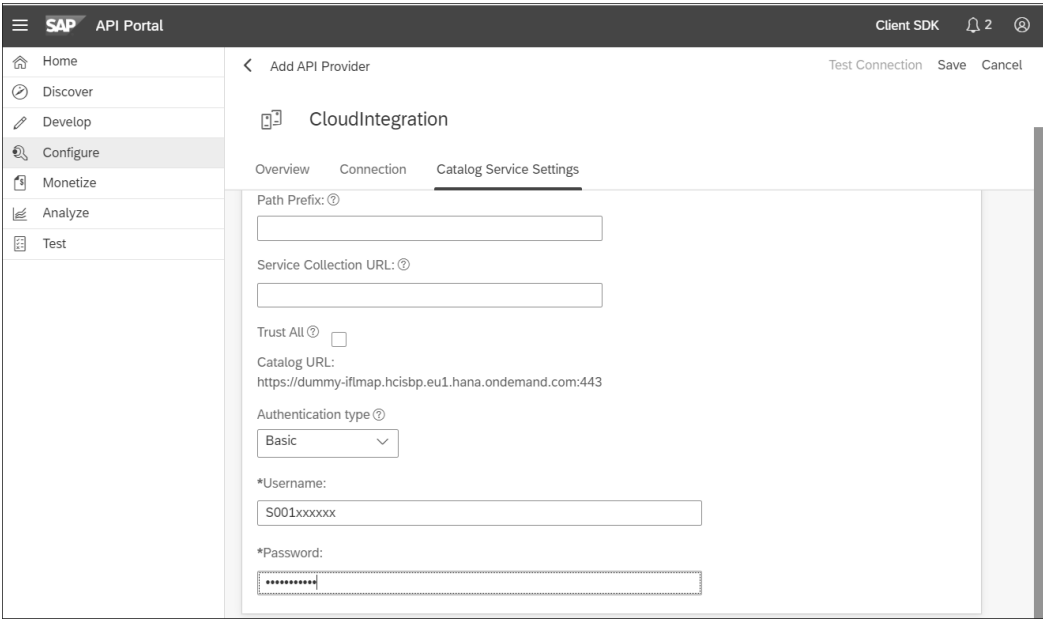


Figure 9.60 Authentication Details for the SAP Cloud Platform Integration Tenant

The connection between SAP API Management and SAP Cloud Platform Integration is now ready. In the next section, we’ll explore how to expose an existing integration flow as a REST API.

9.6.2 Provisioning Application Programming Interfaces

Much could be said about the topic of provisioning APIs through SAP API Management. Our intention isn’t to provide a guide for SAP API Management in this section, but rather to give you a glimpse of what it can do and how it can work in combination with SAP Cloud Platform Integration to create APIs.

To illustrate the provisioning of an API via SAP API Management, let’s once again use our example integration flow built in Chapter 4, Section 4.3. We already used this integration flow in Section 9.3 when exploring the **Script** step (shown earlier in Figure 9.11). This scenario currently exposes a SOAP endpoint. Our goal is to provide this SOAP service as a REST-based API in SAP API Management and apply restrictions to the API by limiting the number of calls per minute. This approach of wrapping an existing service as an API is known as an *API proxy*. REST APIs can accept both JSON or XML as payloads. For simplicity, we’ll stick to XML for our scenario. The final end-to-end scenario is shown in Figure 9.61. Because the integration flow already exists in SAP Cloud Platform Integration, we only need to expose it as an API.

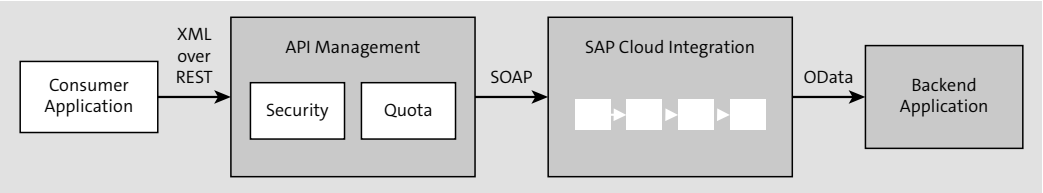


Figure 9.61 End-to-End Overview of Scenario

Let’s recall from Chapter 4, Section 4.3, that the endpoint of the concerned integration flow was followed this format:

`https://<tenant>/cxf/CPI_Book_Demo_OData`

We can now start the provisioning of our API by first navigating to the landing page of SAP API Management, as shown in Figure 9.62. (We described how to get to this page earlier in Section 9.6.1.)

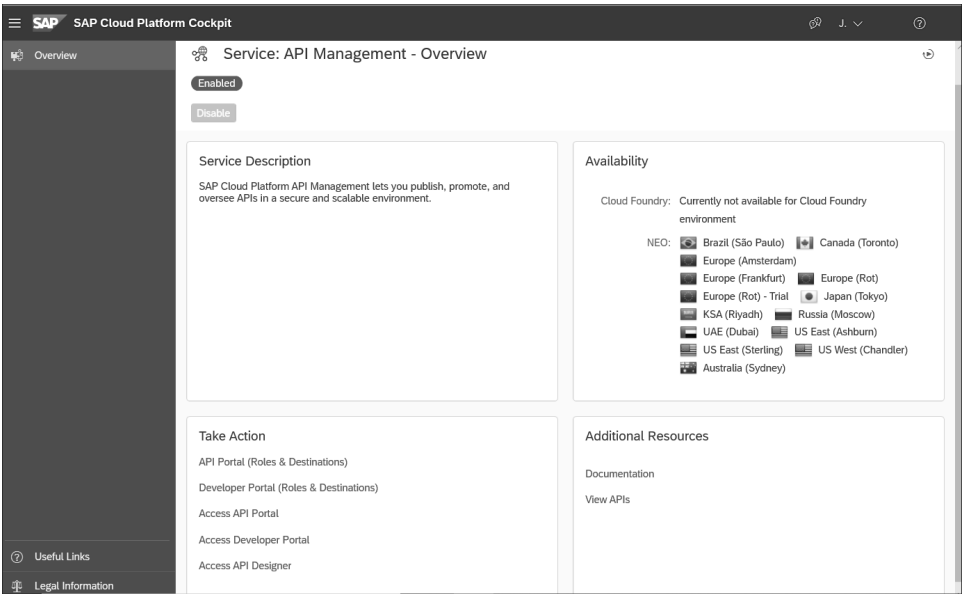


Figure 9.62 SAP API Management: Main Page

As shown in Figure 9.63, the **Take Action** section includes the following options:

■ **API Portal (Roles & Destinations)**

Used to assign roles to users who can create APIs. You can also create destinations to different API providers. A *destination* is an object that contains connection details to a remote system or application. The connection details include the URL of the remote system or service, authentication type, and the user credentials. A destination is defined once and reused throughout the system.

■ **Developer Portal (Roles & Destinations)**

Used to assign roles to users interested in discovering and consuming APIs. (They are also commonly referred to as “developers” from an SAP API Management point of view.) Additionally, from this option, you can create destinations to different API providers.

■ **Access API Portal**

Enables access to various monitoring matrixes about APIs, applications, and products. You can design and create new APIs with this option.

■ **Access Developer Portal**

Enables the developer community to access and subscribe to available APIs in SAP API Management.

■ **Access API Designer**

Enables access to an API editor with rich capabilities for importing existing open APIs, downloading APIs, generating equivalent HTML output views, and validating open API syntax.



Figure 9.63 SAP API Management: Take Action Options

For our discussion, from the page shown in Figure 9.62, click on the **Access API Portal** link. You’ll be redirected to a page similar to the page shown in Figure 9.64.

From this page, you can access monitors and statistics about API performance, traffic, usage, and errors. Furthermore, you can create new APIs, API providers, and applications.

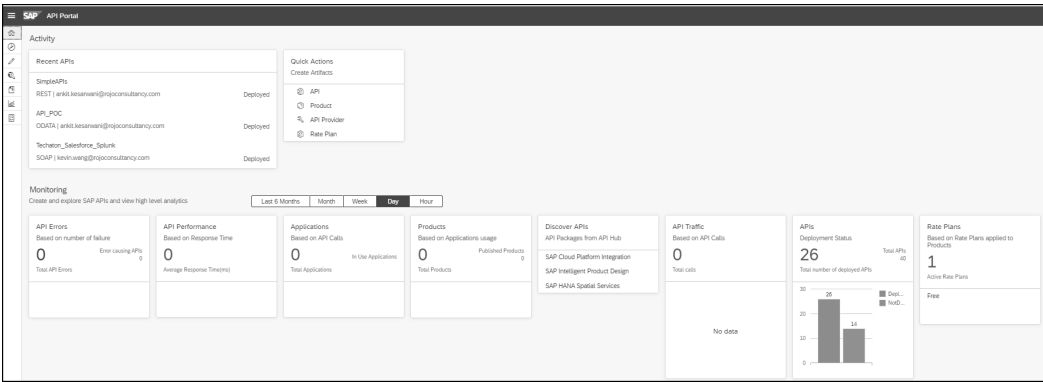



Figure 9.64 API Portal Landing Page

Let’s continue by creating an API by clicking on the **API** icon  , which can be found in the top-right corner of the **Quick Actions** tile, as shown in Figure 9.64. A page will open where you’ll fill in the details about the API proxy. The API proxy must point to the integration flow in SAP Cloud Platform Integration. Specify the details as shown in Figure 9.65 and Figure 9.66. (Note that both figures are parts of the same page.)

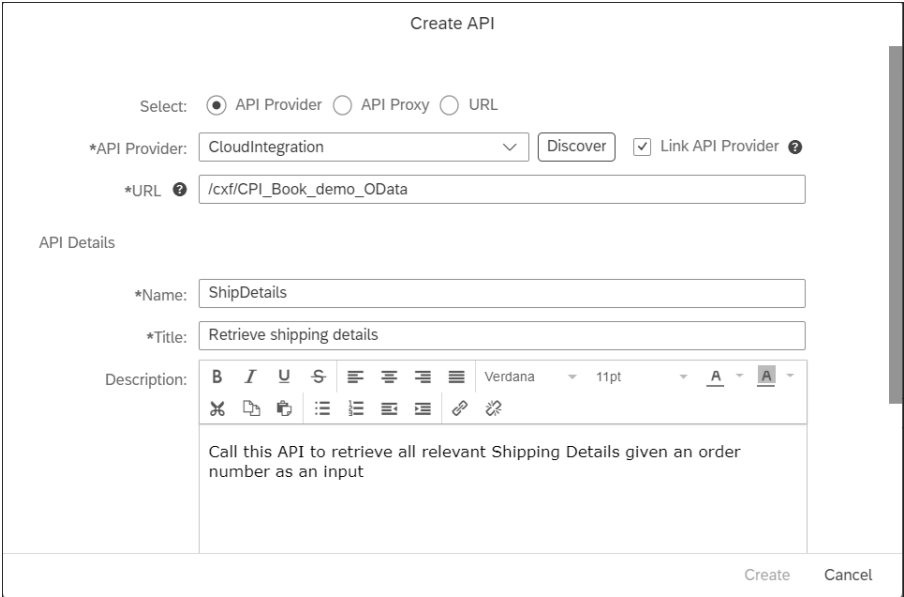


Figure 9.65 Adding Details of the API Proxy to Be Created: Top of Screen

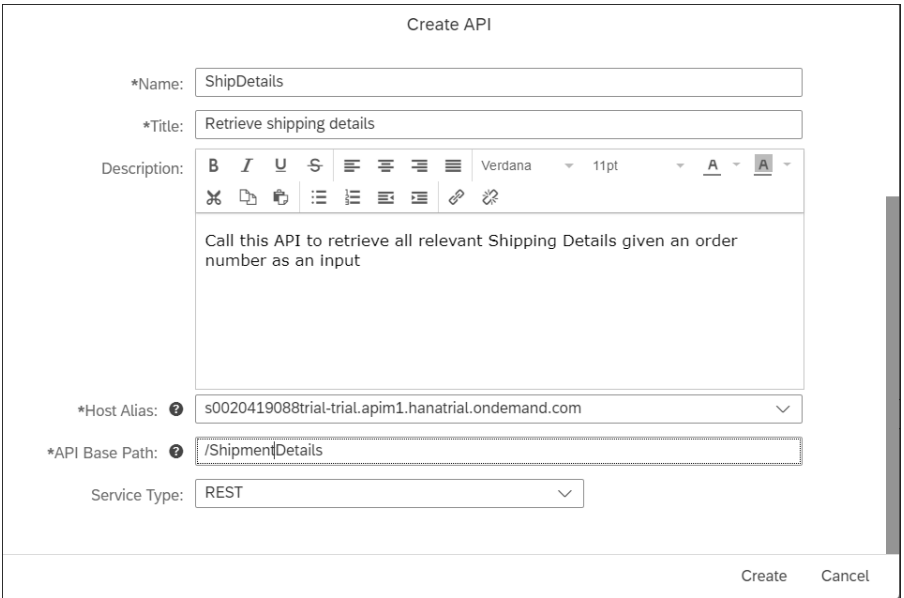


Figure 9.66 Adding Details of the API Proxy to Be Created: Bottom of Screen

Table 9.17 provides descriptions of the fields used in the screen shown in Figure 9.65 and Figure 9.66.

Field	Description
API Provider	Specifies the API provider we created in Section 9.6.1 to connect to the SAP Cloud Platform Integration tenant.
URL	The last part of the integration flow’s endpoint, which can be found from integration artifact monitoring, as discussed in Chapter 4, Section 4.1.
Name	Meaningful name for the API.
Title	Title for the API.
Description	Description for the API.
Host Alias	Automatically populated with the host details of our SAP API Management tenant. Leave this field with its default value.
API Base Path	Specifies the base path to be used as part of the endpoint for the API.
Service Type	Possible values include REST, SOAP, and ODATA.

Table 9.17 API Proxy Fields

As shown in Figure 9.67, you can add a resource by clicking the **Add** button. An API can have multiple resources, and each a resource represents an endpoint.

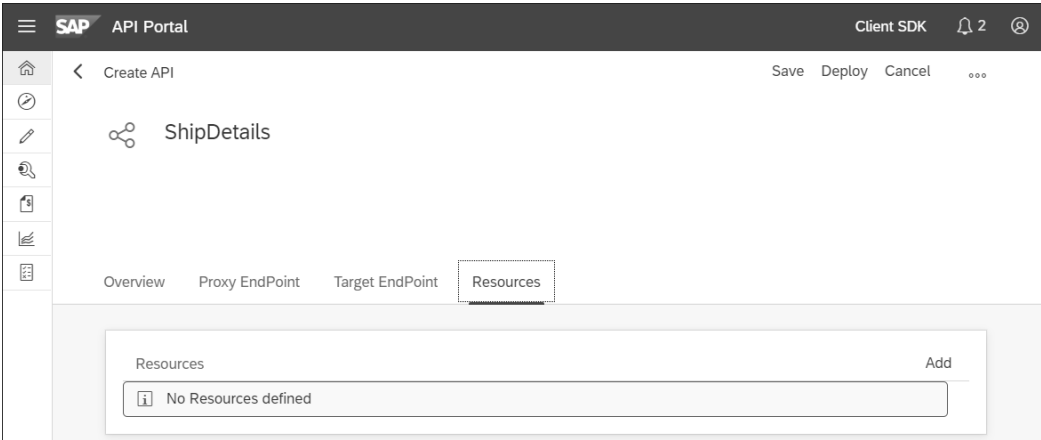


Figure 9.67 Resource Tab of the API Proxy

On the next screen, as shown in Figure 9.68, enter “RetrieveShipment” as the title in the **Tag** field and enter “Shipment” as the path prefix of the resource in the **Path Prefix** field. Remove all other HTTP methods and only select **POST** as the supported HTTP method. Then, specify a description and click the **Add** button.

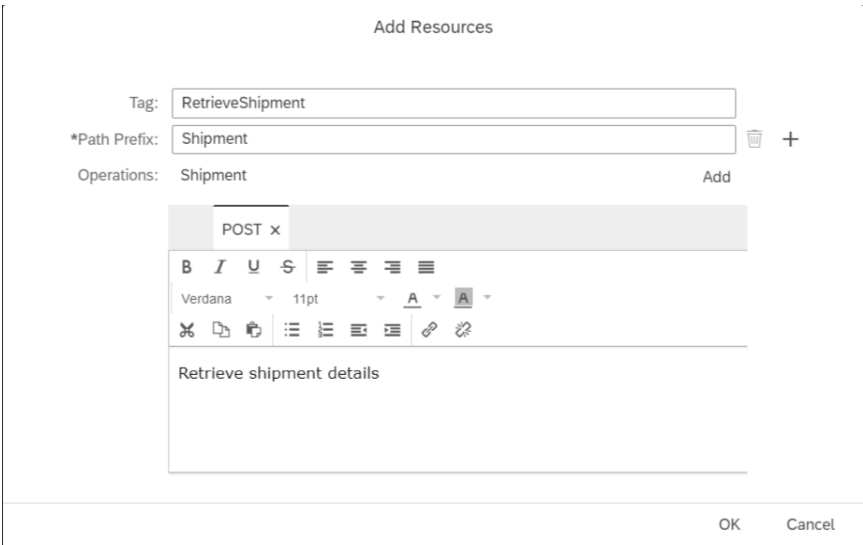


Figure 9.68 Adding a Resource to an API

Let’s now add some policies to our API. An API policy is a module that implements a specific API behavior. API policies are designed to let you add common management capabilities to an API, such as security, rate-limiting, transformation, and mediation. You can access the policy editor by clicking the **Policies** button in the top-right corner of the screen, as shown in Figure 9.69.

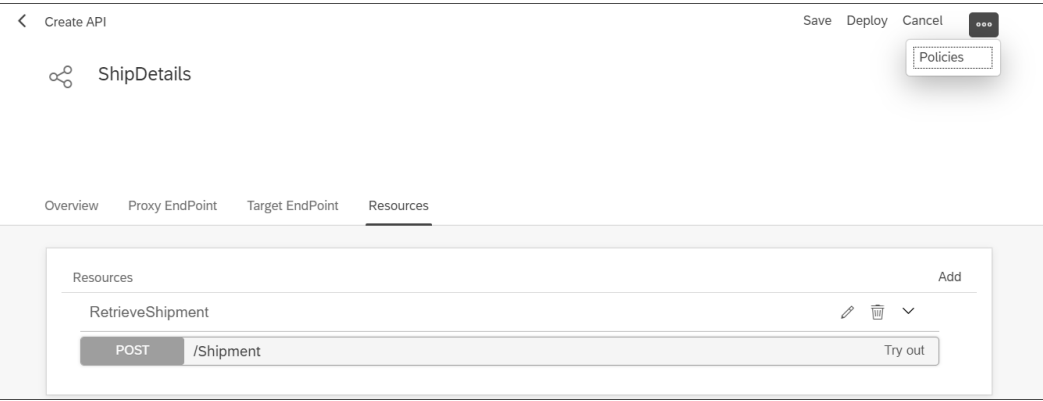


Figure 9.69 Navigating to the Policy Editor

You’re redirected to the policy editor page from which a wide variety of policies can be added to fulfill your requirements, as shown in Figure 9.70.

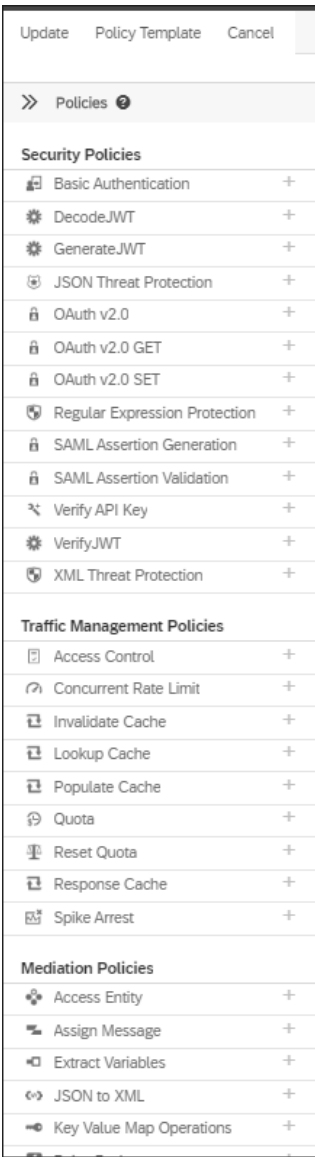


Figure 9.70 An Overview of the Policy Editors

At the time of this writing, the following categories of API policies are included in the policy editor:

- **Security**
Includes various policies aimed at protecting your API. You can add basic authentication; use different versions of OAuth, Security Assertion Markup Language (SAML), and XML thread protection; verify API keys; and so on.

■ **Traffic management**

Helps you regulate your API traffic using techniques such as caching, quotas, access control, concurrent rate limit, and so on.

■ **Mediation**

Actions and scripts, such as for the extraction of variables and conversion of JSON to XML (and vice versa), are bundled in this category.

Note that any number of policies included in this editor can be mixed together and used in any combination to fulfill your desired requirements.

The rectangular image shown in the middle of Figure 9.71 represents a policy flow in SAP API Management. This policy flow defines a processing pipeline and the order of execution of the included policies.

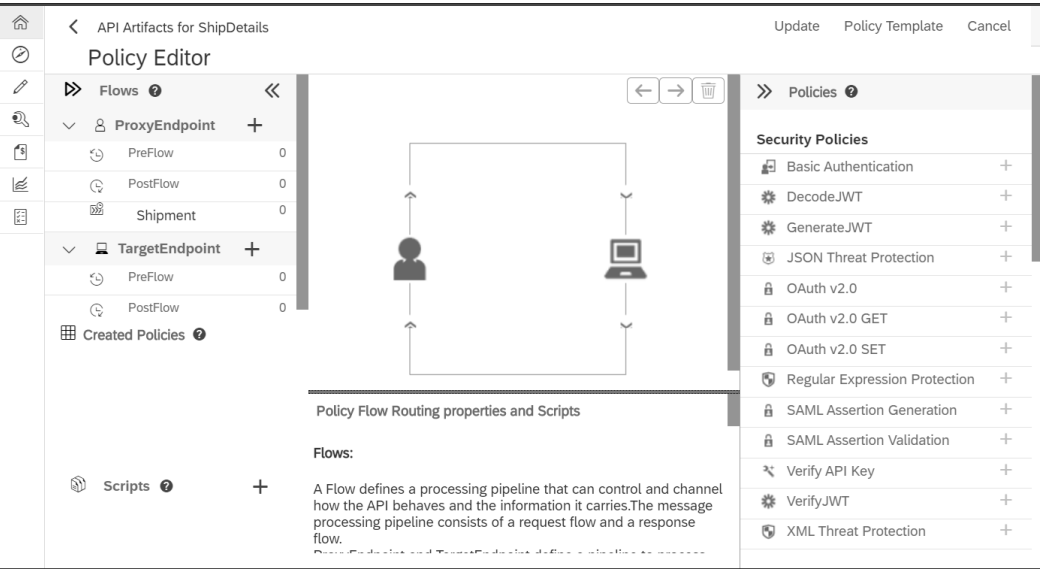


Figure 9.71 API Artifacts View

The flow has two main execution paths: **PreFlow** and **PostFlow**. The **PreFlow** path, the top part of the rectangle, represents the actions to be executed first before the control is passed to the API provider (SAP Cloud Platform Integration, in our case). On the other hand, the **PostFlow** path, the bottom part of the rectangle, represents the actions to be performed after the API provider has been called, that is, the actions to be performed before sending the response back to the API consumer.

For simplicity, let's add a policy to check the quotas on API calls. Assuming that our backend system can only accept a limited number of calls per minute, let's use SAP API Management to limit API consumption to a maximum of two API calls per minute. This approach could a good way to regulate traffic by preventing your backend system from

being flooded with calls. Adding a quota policy in the **PreFlow** path makes sense and prevents calls to SAP Cloud Platform Integration if a quota violation occurs.

Let's start by selecting **PreFlow** in the top-left corner, under the **Flows** section (shown earlier in Figure 9.70). Then, click on the plus icon **+** next to **Quota** on the right panel, under the **Traffic Management Policies** section. You're then presented with a popup window where you'll provide the name of the policy, as shown in Figure 9.72.

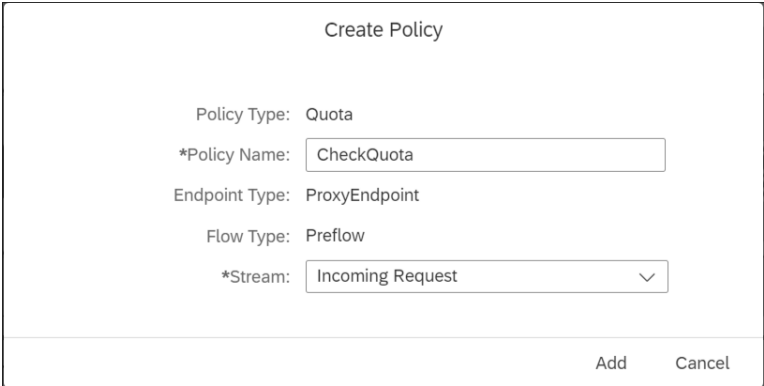


Figure 9.72 Adding the Quota Policy

Besides providing a name for the policy, select the **Incoming Request** value for the **Stream** dropdown, as shown in Figure 9.72. Then, click the **Add** button. On the next screen, specify the maximum number of allowed API calls per minute in the tag element **Allow count**, as shown in Figure 9.73. In our case, we used the value 2 to fulfill our requirement, as described previously.

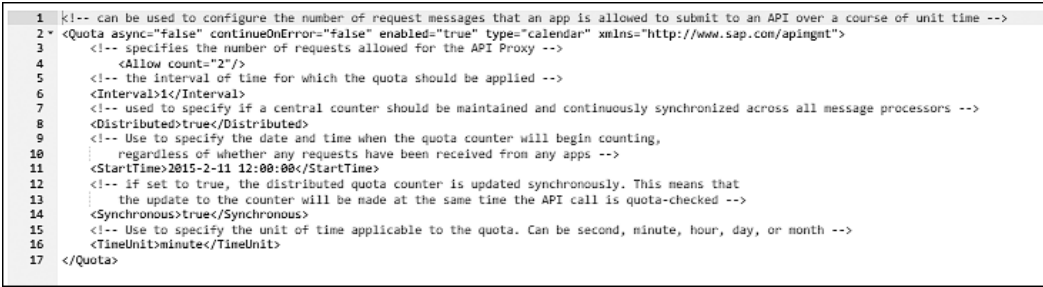


Figure 9.73 Configuration of CheckQuota

Click the **Update** button, located in the top-right corner, as shown in Figure 9.74, to update the API with the added policies.

To find more information about any policy, refer to the SAP Documentation at <https://help.sap.com/viewer/product/CP/Cloud>. Search for "SAP Cloud Platform API Management."

On the next screen, click the **Save** button. Well done! You’re now ready to consume the API, which we’ll do in the next section.

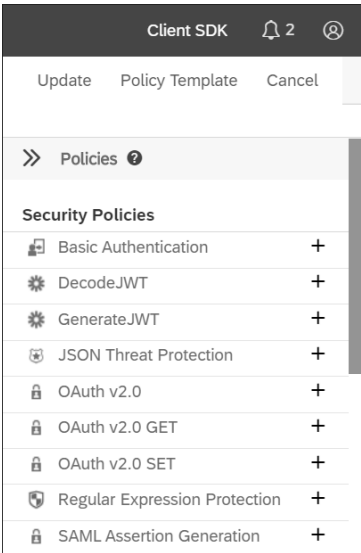



Figure 9.74 Updating the API with the Flow of Policies

9.6.3 Consuming the Application Programming Interface

Now that the API is ready, you can test it using any REST client of your choice (e.g., Postman). You can also perform a test directly in SAP API Management by navigating to the test tool via the **Test** icon , at the bottom of menu on the left, as shown in Figure 9.75.

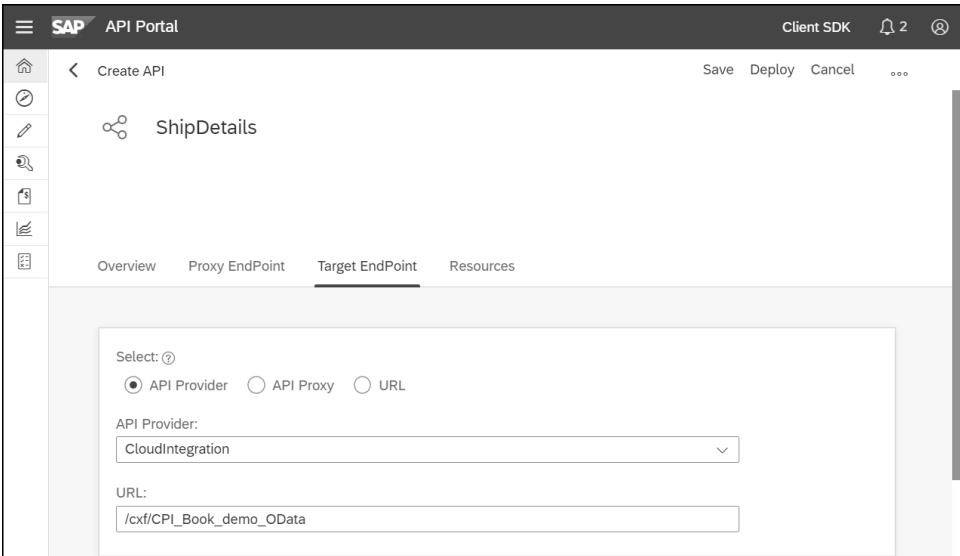


Figure 9.75 Overview of the Created API

You’re then redirected to a new test page where you should select **ShipDetails** on the left side of the screen shown in Figure 9.76. Note that, by default, an endpoint ending with **/SWAGGER_JSON** is selected, as shown in Figure 9.76. You’ll need to select the other endpoint from the dropdown list. The correct endpoint will end with the prefix used while creating the resource (in our scenario, **/Shipment**).

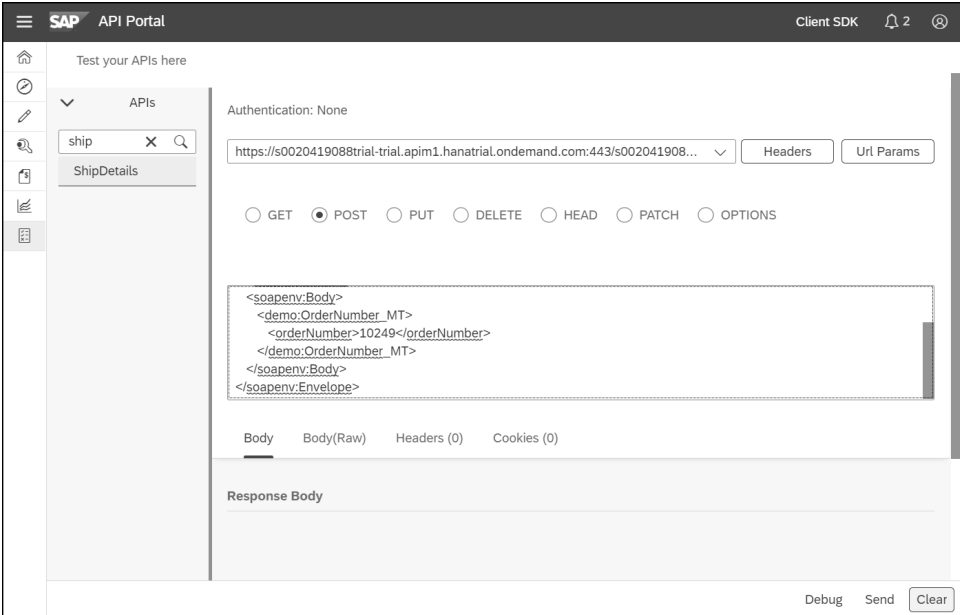


Figure 9.76 Setting Up the API Test Tool

Note

If you’re using an external REST client such as Postman, you’ll need an endpoint to post the API call to. The endpoint of the newly created API can be obtained via the drop-down menu, as shown in Figure 9.76.

After selecting the correct endpoint, specify the HTTP method as **POST** by selecting its radio button, provide the desired request XML payload, and change the authentication method to basic authentication. The authentication can be changed by clicking the **Authentication** link in the top-left corner of the screen shown in Figure 9.76.

You can now trigger the call by clicking on the **Send** button at the bottom-right corner of the screen. In the background, SAP API Management performs a check to validate whether we’re still within our quota limits. Because this message is the first message, the call is accepted and sent to SAP Cloud Platform Integration, which in turn returns a valid response, as shown in the **Response Body** section shown in Figure 9.77.

Quickly perform the same call two more times to exceed our quota limit. You'll then see a quota violation error, as shown in Figure 9.78. This error indicates that we've exceeded the quota of two calls within the same minute.

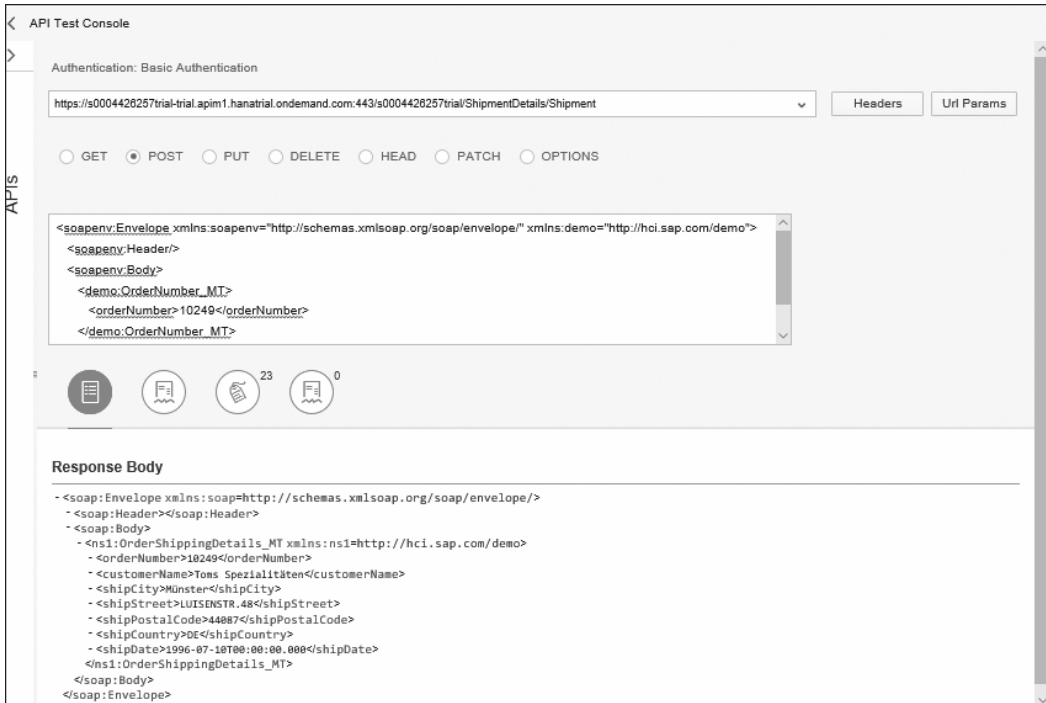


Figure 9.77 The Test Result of Our API Call

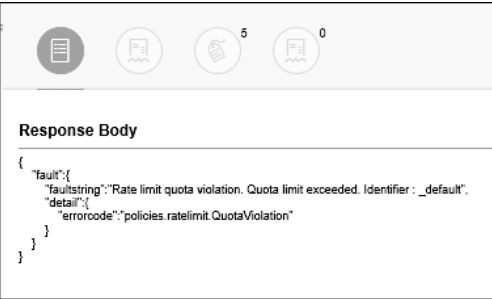


Figure 9.78 Response When Quota Is Exceeded

Congratulations, you've successfully provided, called, and consumed the API. Be aware that our scenario was quite simplistic. In a real-life scenario, you should consider the following additional aspects:

■ Authentication in SAP API Management

By default, the API exposed can be consumed without authentication. If you want to protect the API, you'll need to add the relevant policy (e.g., OAuth, API key, etc.).

■ Authentication in SAP Cloud Platform Integration

In our scenario, we're simply passing along to SAP Cloud Platform Integration whatever header was provided during the API call. Thus, the user name/password details used while calling the API are also forwarded and used for authentication in SAP Cloud Platform Integration. You could decide to use the basic authentication security policy to provide the logon details for SAP Cloud Platform Integration.

■ Payload

Note that the sample request message shown in Figure 9.76 includes the entire SOAP message. Given that we're exposing a REST API, we could decide to only use the SOAP body as input. This option, however, will require you to perform some logic to extract the relevant data from the incoming message and construct the SOAP message to send to SAP Cloud Platform Integration. This step is necessary because the service that we're consuming in SAP Cloud Platform Integration is of type SOAP.

■ Error message

Note that the error response returned by the API when the quota was exceeded (as shown in Figure 9.78) is in JSON format. In a real-life scenario, you'll want to convert the response to XML before returning this data to the consumer. (Hint: consider the JSON2XML policy.)

This chapter didn't tackle these points, which are beyond the scope of this chapter. For more insights on the subject of SAP API Management, refer to the book *SAP API Management* (SAP PRESS, 2019) written by Carsten Bönner, Harsh Jegadeesan, Divya Mary, and Shilpa Vij (www.sap-press.com/4928).

You now understand how to use SAP API Management to wrap services available in SAP Cloud Platform Integration and expose them as APIs, as well as how to enforce different policies.

9.7 Summary

This chapter introduced you to the API-related capabilities and features of SAP Cloud Platform Integration. An overview of Java-based APIs was provided, and you also learned how to use APIs in UDFs. This chapter then explored a number of OData APIs available in SAP Cloud Platform Integration and walked you through some examples to illustrate their usage.

Finally, this chapter explored how to use SAP Cloud Platform Integration, in combination with SAP API Management, to provision APIs. Even though this chapter wasn't intended to be a chapter on SAP API Management, you also learned how to add policies to APIs through a scenario.

In the next chapter, we'll explore SAP Cloud Platform Open Connectors.

Contents

Foreword by Gunther Rothermel	17
Preface	21
Acknowledgments	27

1 Introduction to SAP Cloud Platform Integration 31

1.1 The Role of SAP Cloud Platform Integration in a Cloud-Based Strategy	33
1.2 Use Cases	36
1.2.1 Point-to-Point versus Mediated Communication	36
1.2.2 Message-Based Process Integration	37
1.2.3 Cloud-to-Cloud Integration	38
1.2.4 Cloud-to-On-Premise Integration	39
1.2.5 On-Premise-to-On-Premise Integration	40
1.2.6 Hybrid Usage of Cloud and On-Premise Integration Solutions	41
1.2.7 Usage in Different Cloud Environments	42
1.3 Capabilities	44
1.3.1 Integration Platform-as-a-Service	44
1.3.2 Message Processing Step Types (Integration Capabilities)	45
1.3.3 Connectivity Options	46
1.3.4 Prepackaged Integration Content	49
1.3.5 Security Features	50
1.3.6 High Availability	51
1.3.7 Integration Design and Monitoring Tools	51
1.4 Editions	51
1.5 Summary	53

2 Getting Started 55

2.1 Architecture Overview	55
2.1.1 Containerized and Clustered Integration Platform	56
2.1.2 Basic Constituents of the Integration Platform	60
2.1.3 Architecture Including User Access and Data Storage Areas	62
2.1.4 Secure Communication	68
2.1.5 Implementation of Message Flows	68

2.1.6	Architecture Summary	72
2.1.7	Environment-Specific Aspects of the Architecture	75
2.2	Tools and Processes	76
2.2.1	Tools	76
2.2.2	Processes	83
2.3	Running Your First Integration Scenario	84
2.3.1	Demo Scenario and Landscape	84
2.3.2	Prerequisites	86
2.3.3	Setting Up the Landscape and the Technical Connections	86
2.3.4	Developing the Integration Flow	88
2.3.5	Creating and Deploying a User Credentials Artifact	113
2.3.6	Importing Certificates Required by the Mail Server into Keystore	117
2.4	Summary	123
3	SAP Integration Content Catalog	125
3.1	Introduction to the SAP Integration Content Catalog	125
3.2	Terms and Conditions of Using Prepackaged Integration Content	128
3.2.1	Quick Configure versus Content Edit	129
3.2.2	Notify about Update (Manual Updates)	130
3.2.3	Automatic Updates	131
3.3	Consuming Prepackaged Content	132
3.3.1	Searching the SAP Integration Content Catalog	132
3.3.2	Importing Prepackaged Integration Content	137
3.3.3	Modifying or Configuring the Integration Package	138
3.3.4	Deploy Content	145
3.4	Prepackaged Content Provided by SAP	145
3.4.1	Content for SAP SuccessFactors	146
3.4.2	Content for SAP Cloud for Customer	148
3.4.3	Content for Integrating with SAP C/4HANA	149
3.4.4	Content for Integrating with the Ariba Network	152
3.4.5	Content for Globalization Scenarios	153
3.4.6	Content for ELSTER Integration	154
3.5	Creating Your Own Content Package	157
3.6	Summary	160

4	Basic Integration Scenarios	163
4.1	Working with SAP Cloud Platform Integration’s Data Model	163
4.1.1	Message Processing: The Apache Camel Framework	165
4.1.2	Exercise: Working with Apache Camel’s Message Model	168
4.1.3	Connecting and Configuring a Sender with an Integration Flow	170
4.1.4	Adding and Configuring Steps in the Integration Flow	173
4.1.5	Checking Configuration Using the Problems Tab	177
4.1.6	Running the Integration Flow	179
4.1.7	Troubleshooting	184
4.2	Using Externalization to Enable Easy Reuse of Integration Flows	185
4.2.1	Externalizing	186
4.2.2	Configuring and Running the Scenario	190
4.3	Content Enrichment by Invoking an OData Service	192
4.3.1	The Target Scenario	193
4.3.2	Invoking an OData Service	194
4.3.3	Configuring the OData Connection	196
4.3.4	Creating the Resource Path Using the Query Editor	198
4.3.5	Using the Content Enricher Step	203
4.4	Working with Mappings	208
4.4.1	The Scenario	209
4.4.2	Adding and Using Resources via the Resources Tab	211
4.4.3	Applying the Mapping Step in the Message Processing Chain	216
4.4.4	Using Value Mappings to Enhance Your Scenario	223
4.5	Defining and Provisioning an OData Service	229
4.5.1	The Target Scenario	229
4.5.2	Providing an OData Service	229
4.6	Working with an Aggregator	241
4.6.1	Sample Scenario	242
4.6.2	Sending Messages via SoapUI	245
4.7	Summary	251
5	Advanced Integration Scenarios	253
5.1	Message Routing	253
5.1.1	The Scenario	254
5.1.2	Configuration of the Content-Based Router	256
5.1.3	Running the Content-Based Router Scenario	260

5.2	Working with Lists	262
5.2.1	The Scenario	263
5.2.2	Configuring the Integration Flow	265
5.2.3	Running the Integration Flow	275
5.2.4	Enriching Individual Messages with Additional Data	279
5.3	Asynchronous Message Handling	281
5.3.1	Synchronous versus Asynchronous Communication from the SAP Cloud Platform Integration Perspective	283
5.3.2	Adding an Asynchronous Receiver	293
5.3.3	Routing a Message to Multiple Receivers Using the Multicast Pattern	297
5.4	Reliable Messaging Using the JMS Adapter	304
5.4.1	Asynchronous Decoupling of Inbound Communication	305
5.4.2	Configuring Retry for Multiple Receivers	316
5.4.3	Configuring Explicit Retry with Alternative Processing	323
5.5	Using Event-Driven Messaging	331
5.5.1	Configuring a Publish-Subscribe Scenario	332
5.5.2	Reading Business Details from Providers	346
5.5.3	Sending Events to SAP Cloud Platform Enterprise Messaging	349
5.6	Summary	354
6	Special Topics in Integration Development	355
6.1	Timer-Based Message Transfers	356
6.1.1	The Scenario	356
6.1.2	Configuring a Timer-Based Integration Flow	357
6.1.3	Running the Integration Flow	361
6.2	Using Dynamic Configuration via Headers or Properties	362
6.2.1	An Integration Flow with a Dynamically Configured Attribute	364
6.2.2	Monitoring Dynamically Configured Attributes at Runtime	369
6.2.3	Using Predefined Headers and Properties to Retrieve Specific Data Provided by the Integration Framework	374
6.3	Structuring Large Integration Flows Using Local Processes	381
6.3.1	Managing Complexity through Modularization	381
6.3.2	Developing an Integration Flow with a Local Integration Process	382
6.3.3	Using Exception Subprocesses	389
6.4	Connecting Integration Flows Using the ProcessDirect Adapter	395
6.4.1	Use Cases for the ProcessDirect Adapter	397

6.4.2	A Simple Example	398
6.4.3	Using Variables to Share Data between Different Integration Flows	403
6.4.4	Dynamic Endpoint Configuration with the ProcessDirect Adapter	406
6.5	Connecting to a Database Using the JDBC Adapter	412
6.5.1	JDBC Adapter Concepts	413
6.5.2	Setting Up a Database System	414
6.5.3	Setting Up an Example Scenario	416
6.6	Versioning and Migration of Integration Flows	420
6.6.1	Integration Flow Component Versions	421
6.6.2	Upgrading an Integration Flow Component	423
6.6.3	Adapting Integration Content for SAP Process Orchestration	426
6.7	Simulation of Integration Flow Processing	433
6.8	Transporting Integration Packages to Another Tenant	441
6.8.1	Manually Transporting Integration Packages	441
6.8.2	Transporting Integration Packages Using CTS+	442
6.8.3	Transporting Integration Packages Using the Cloud-Based Transport Management Service	442
6.9	Using the Adapter Development Kit (ADK)	445
6.9.1	Overview	445
6.9.2	Installing the Adapter Development Kit	446
6.9.3	Developing a Sample Adapter (Neo and Cloud Foundry)	449
6.9.4	Deploying the Adapter (Neo)	455
6.9.5	Deploying the Adapter (Cloud Foundry)	457
6.9.6	Testing the New Adapter	461
6.10	Guidelines for Integration Flow Development	463
6.10.1	Running an Integration Flow Under Well-Defined Boundary Conditions	465
6.10.2	Relaxing Dependencies to External Components	466
6.10.3	Keeping Readability in Mind	466
6.10.4	Handling Errors Gracefully	467
6.10.5	Applying the Highest Security Standards	467
6.10.6	Additional Best Practices	467
6.11	Summary	468

7	B2B Integration with the SAP Cloud Platform Integration Suite	471
7.1	B2B Capabilities in the SAP Cloud Platform Integration Suite: Overview ...	472
7.2	Defining Interfaces and Mappings in the SAP Cloud Platform Integration Advisor	474
7.2.1	Creating Message Implementation Guidelines	476
7.2.2	Configuring Mapping Guidelines	488
7.2.3	Generating the Runtime Content	490
7.3	Configuring a B2B Scenario with AS2 Sender and IDoc Receiver Adapters	490
7.3.1	Creating an Integration Flow Using a Template	491
7.3.2	Configuring the AS2 Sender Channel to Receive EDI Messages	498
7.3.3	Adding an IDoc Receiver	504
7.3.4	Configuring Acknowledgment Handling	508
7.4	Using the Partner Directory for Partner-Specific Configuration Data	518
7.4.1	Concept of the Partner Directory	518
7.4.2	Using a Receiver Endpoint URL Dynamically in the Integration Flow	520
7.4.3	Storing the Partner-Specific Endpoint URL in the Partner Directory	523
7.5	Summary	532
8	SAP Cloud Platform Integration Operations	533
8.1	Operations: Overview	533
8.2	Monitoring Integration Content and Message Processing	535
8.2.1	Managing Integration Content	538
8.2.2	Log Configuration	541
8.2.3	Monitoring Message Processing	543
8.2.4	Managing Tiles	557
8.3	Managing Security	560
8.3.1	Maintaining Security Material	562
8.3.2	Managing the Keystore	565
8.3.3	Defining User Roles	573
8.3.4	Maintaining Certificate-to-User Mappings	575
8.3.5	Defining Access Policies	577
8.3.6	Managing JDBC Data Sources	581
8.3.7	Testing Outbound Connectivity	584

8.4	Managing Temporary Data	600
8.4.1	Monitoring Data Stores	600
8.4.2	Monitoring Variables	603
8.4.3	Maintaining Message Queues	605
8.4.4	Maintaining Number Ranges	614
8.5	Accessing Logs	616
8.5.1	Monitoring Audit Logs	616
8.5.2	Checking System Log Files	618
8.6	Managing Locks	620
8.7	Summary	624
9	Application Programming Interfaces	625
9.1	Introduction	625
9.2	Java APIs Provided by SAP Cloud Platform Integration	626
9.3	Using the Java API in a User-Defined Function	628
9.4	Using the Script Step	633
9.4.1	Target Scenario	634
9.4.2	Enhancing the Integration Flow	634
9.5	OData API	637
9.5.1	SAP API Business Hub	641
9.5.2	Cross-Site Request Forgery Token Handling for Neo	655
9.5.3	Monitoring Message Flows Using the API	658
9.5.4	Managing Deployed Integration Content Using the API	664
9.5.5	Managing Log Files Using the APIs	666
9.5.6	Managing Message Store Entries Using APIs	668
9.5.7	Managing Security Material Using the API	671
9.5.8	Managing the Partner Directory Using the API	672
9.6	Using SAP Cloud Platform Integration with SAP Cloud Platform API Management	673
9.6.1	Establishing a Connection between SAP Cloud Platform Integration and SAP API Management	676
9.6.2	Provisioning Application Programming Interfaces	679
9.6.3	Consuming the Application Programming Interface	688
9.7	Summary	691

10 Integration with SAP Cloud Platform Open Connectors	693
10.1 Introduction	693
10.2 Connectors Catalog	695
10.2.1 Overview	698
10.2.2 Information	700
10.2.3 Setup	701
10.2.4 Resources	702
10.2.5 Validation	703
10.2.6 API Docs	704
10.3 Understanding Connectors	708
10.3.1 Authenticate Connections with API Providers	709
10.3.2 Inspect Authenticated Connector Instances	712
10.3.3 Testing in the API Docs	714
10.4 Understanding Common Resources	718
10.4.1 Creating a Common Resource	720
10.4.2 Cloning Common Resources	722
10.4.3 Transforming Resources	724
10.5 Using SAP Cloud Platform Integration with SAP Cloud Platform Open Connectors	727
10.5.1 Scenario	727
10.5.2 The Solution	728
10.6 Summary	737
11 SAP Cloud Platform Integration Security	739
11.1 Technical System Landscape	740
11.1.1 Architecture	740
11.1.2 Network Infrastructure	743
11.1.3 Data Storage Security	745
11.1.4 Data Protection and Privacy	746
11.1.5 Physical Data Security	748
11.2 Processes	749
11.2.1 Software Development Process	749
11.2.2 Operating the Cloud Infrastructure and Providing and Updating the Software	750

11.2.3 Setting Up Secure Connections between the Tenant and Remote Systems	751
11.3 User Administration and Authorization	752
11.3.1 Technical Aspects of User Management	753
11.3.2 Personas, Roles, and Permissions	754
11.3.3 Managing Users and Authorizations for an SAP Cloud Platform Integration Subaccount	755
11.3.4 Authorization on Package and Integration Flow Level	766
11.4 Data and Data Flow Security	772
11.4.1 Basic Cryptography in a Nutshell	773
11.4.2 Transport-Level Security Options	779
11.4.3 Authentication and Authorization	781
11.4.4 Securely Connecting a Customer System to SAP Cloud Platform Integration (through HTTPS)	795
11.4.5 Setting Up a Scenario Using OAuth with the Twitter Adapter	806
11.4.6 Message-Level Security Options	815
11.4.7 Designing Message-Level Security Options in an Integration Flow	818
11.5 Keystore Management	839
11.5.1 Using X.509 Security Material for SAP Cloud Platform Integration	839
11.5.2 Managing Security Material in the Tenant Keystore	843
11.5.3 Managing the Lifecycle of Keys Provided by SAP	847
11.6 Summary	852
12 Productive Scenarios Using SAP Cloud Platform Integration	855
12.1 Integration of SAP Cloud for Customer and SAP ERP	855
12.1.1 Technical Landscape	856
12.1.2 Example Adapter Configurations	857
12.2 Integration of SAP Cloud for Customer with SAP S/4HANA Cloud	860
12.3 Integration of SAP Marketing Cloud and Various Applications	861
12.4 Integration of SAP SuccessFactors and SAP ERP	862
12.4.1 Technical Landscape	863
12.4.2 SAP SuccessFactors Adapter	864
12.5 Integration of SAP Applications with the Ariba Network	866
12.5.1 Technical Landscape	868

12.6	Integration with German Tax Authorities Using the ELSTER Adapter	868
12.7	Summary	871

13 Summary and Outlook 873

13.1	Integration Content Design	874
13.2	Operations and Monitoring	875
13.3	Connectivity	876
13.4	B2B Integration	877
13.4.1	B2B Monitoring	878
13.4.2	Trading Partner Management	878
13.5	Migration from Neo to Cloud Foundry Environment	879
13.6	Hybrid Deployments on Cloud and On-Premise Infrastructures	879
13.7	Summary	880

Appendices 881

A	Abbreviations	883
B	Literature	889
C	The Authors	893

Index	895
-------------	-----

Index

A

Absolute path	269, 273, 276
Access logs	616
Access policy	561, 577, 766
<i>definition</i>	769
<i>monitor</i>	768
Account	713, 720, 721
Account member	756
Accredited Standards Committee X12 (ASC X12)	471, 476
Acknowledgment	508, 509, 514
Activities	695
Adapter	148, 153
<i>deployment in Cloud Foundry</i>	457
<i>deployment in Neo</i>	455
<i>development</i>	445, 449
<i>Elektronische Steuererklärung (ELSTER)</i>	868
<i>Facebook adapter</i>	47
<i>HTTP adapter</i>	47
<i>HTTPS sender</i>	95
<i>IDoc (SOAP) adapter</i>	47
<i>Java Database Connectivity (JDBC)</i>	412
<i>Java Message Service (JMS) adapter</i>	374
<i>JDBC</i>	67
<i>mail adapter</i>	47
<i>mail receiver adapter</i>	108
<i>mail sender adapter</i>	122
<i>OData adapter</i>	47
<i>ProcessDirect</i>	395
<i>receiver</i>	792
<i>SAP Ariba adapter</i>	47
<i>SAP SuccessFactors adapter</i>	47
<i>SFTP</i>	123
<i>SFTP adapter</i>	47
<i>SOAP adapter</i>	47
<i>Twitter adapter</i>	47, 806
<i>type</i>	109, 195, 295
Adapter development <i>Maven support</i>	452
Adapter Development Kit (ADK)	445–447
Advanced Message Queuing Protocol (AMQP)	334, 585, 598, 780
<i>adapter</i>	331, 341, 342
<i>receiver adapter</i>	333, 349, 351
<i>sender adapter</i>	333, 343
Aggregation algorithm	273
AK3	515
AK4	515
AK5	514, 515
Alias	566, 571, 572
Alibaba Cloud	751
Amazon Web Services (AWS)	751
Apache Camel	38, 70, 165–167, 176, 177, 184, 200, 209, 254, 262, 283, 286, 381, 445
<i>Camel route</i>	71
API Docs	697, 698, 704
Applicability Statement 2 (AS2) <i>adapter</i>	473, 498, 605, 621
<i>receiver adapter</i>	510
<i>sender adapter</i>	498, 499
Applicability Statement 4 (AS4) <i>adapter</i>	473, 600, 606, 621
Application edition	52
Application ID	544
Application identifier	759
Ariba Network	50, 149, 152, 153, 866
<i>content</i>	152
Artifact	126, 129, 130, 134–137, 139, 141, 142, 157, 159, 544
<i>details</i>	541
<i>references</i>	579
<i>STATUS</i>	539
<i>TYPE</i>	539
<i>user credentials</i>	113
Asymmetric key technologies	774
Asynchronous communication	251
Asynchronous decoupling	305
Asynchronous message handling	167, 281, 282
Asynchronous receiver	293, 304
Atom	201, 210
Attachment	166
Audit log	616, 747
Authentication	197, 701, 781
<i>basic</i>	782, 788, 789, 791, 793
<i>client certificate</i>	782, 789, 793, 796
<i>inbound</i>	787, 788
<i>OAuth</i>	782, 790, 791, 794
<i>outbound</i>	792
<i>type</i>	172
Authorization	96, 266, 766, 781
<i>client certificate</i>	788, 803
<i>group</i>	754
<i>server</i>	790

Authorization (Cont.)

- user role* 266, 787

Authorization group 638

- business expert* 638
- integration developer* 638
- tenant administrator* 638

Avoiding hardcoded values 186

B

B2B integration 35

B2B scenario 74

Backup 567, 569

Basic authentication 111, 172, 182

Body 111, 174–176, 255, 257, 286, 707

Branch 554, 555

Bug fixes 535

Bulk message 272

Business Process Model and Notation (BPMN) 69, 163–165, 255, 297

Business-to-business (B2B) 471

- capabilities* 472
- integration* 471
- scenario* 490

C

Call 303

Camel component 446

Cancelled 544

Capacity 613

Certificate 560, 561, 565, 777

- chain* 569, 777
- distinguished name* 778, 846

Certificate type

- X.509* 778

Certificate-to-user mapping 561, 562, 575, 789

Change and Transport System (CTS+) 442

- enhanced* 442

Channel 171

Check mail addresses 594

Child process 386, 388

ChildCount 554, 556

ChildrenCounter 554

Chunking queue 607

Client certificate 197

clientid 336

clientsecret 336

Cloud 39

- computing benefits* 33
- on-premise integration* 40

Cloud (Cont.)

- platform* 534
- strategy* 35

Cloud Foundry 58, 75, 748, 750

Cloud infrastructure 750

- provider* 750

CloudEvents 339

Comma-separated values (CSV) 210

Common resources 695, 718

Communication 35, 742

- inbound* 819
- outbound* 819

Completed 544, 559

- messages* 557

Configurations 701

Configure-only 129, 131, 138, 142, 143

Connection 83

- inbound* 762
- type* 171

Connectivity 46

- testing* 82

Connectivity test 562

- tile* 584

Connectivity test tool 799, 805

Connectors 695

- build new* 696

Connectors catalog 695

Consumer 332, 613

Consumer integration flow 396

Container 56

Content

- Ariba Network* 152
- globalization scenarios* 153, 154
- SAP Cloud for Customer* 148
- SAP SuccessFactors* 146

Content edit 128, 129, 138

- conditions* 129

Content enricher 192, 203, 207

Content for globalization scenarios 146

Content modifier 164, 174–176, 193, 434

Content publisher 126

Content reviewer 126

Content-based

- router* 38
- routing* 251, 253

Content-based routing (CBR) 253–256, 260, 262

ContextName 554

Correlation ID 544, 546, 554

Country 133

CPI default Trace 619

CPU 534

Create user credentials artifact 111

Credential name 111, 116

Credentials 560

Cross-site request forgery (CSRF) 672

- X-CSRF tokens* 525

Cryptography 773

CSRF 525

cURL 730

Custom adapter 355

Custom resources 719

customer relationship management (CRM) 855

Customer workspace 135, 137, 145

cXML 477

D

Data flow security 772

Data flows 139

Data integration 159

Data integrity 774

Data model 163, 168

Data protection 746

- dialog user access* 747
- message content* 745
- monitoring data* 464, 747

Data storage 413, 742

Data store 66, 82, 84, 305, 600

- GET* 306
- monitoring* 105
- operations* 96
- parameters* 99
- SELECT* 306
- WRITE* 306
- write step* 813

Database 45, 412

- access token* 416

Database ID 583

Database system

- configuration* 414
- SAP ASE* 413
- SAP HANA* 413

Dead-letter queue 313, 621

- handling* 621
- option* 621

Debug 542

Default gate 255

Default route 258, 260

DELETE 700

Demilitarized zone (DMZ) 744

Deployment 140, 179, 563

- fast* 33

Design 127, 137, 142, 157

- view* 168
- workspace* 142

Design guidelines 463

- apply security* 467
- handle errors* 467

Design, Monitoring, and Operation

- microservices* 743

Digest 816

Digital certificate 777

Digital encryption 50

Digital signature 50, 816

Discover 127, 133

Distinguished name (DN)

- issuer* 846
- subject* 846

Documents 134, 159

Download 565

Download artifact 540

Download logs 553

Dynamic configuration 362, 406

Dynamic expression

- exception message* 392

E

EANCOM 476

Eclipse 83, 446–448

- Oxygen* 447

Edit mode 170, 669

eDocument Framework 154

- Chile* 154
- Hungary* 154
- Italy* 154
- Peru* 154
- Spain* 154

Electronic Data Interchange (EDI)

- EDI to XML Converter* 473, 494
- extractor* 473
- splitter* 267, 473, 493, 494, 508, 509
- validator* 473

Elektronische Steuererklärung (ELSTER) 868

Email 166

Email receiver 113, 293, 294

Empty start event 384

Encryption 500, 773

End event 164

Endpoint 179, 540

Endpoint URL 180

Enricher pattern 264

Enterprise integration patterns 38, 45, 254

- example* 38

Enterprise Services Builder 210, 264

Enterprise Services Repository (ESR) 209, 210, 220

Entity Data Model Designer (EDMX) 201

Error 554, 563

analysis 185

queue 607

Error end event 394

Error handling 389, 467

response 394

Escalated 544, 558, 559

European Union General Data Protection

 Regulation (GDPR) 748

Evaluation condition 257

Evaluation sequence 262

Event 332, 702

Event bus 332

Event hub 332, 334

Event-based messaging 331

Exception 167

Exception subprocesses 389

Exchange 111, 166, 167, 174, 175, 177, 283, 286, 291, 382, 389

Exchange ID 167

Exchange property 167, 168, 174, 175, 177, 183, 363, 374, 466

Exclusive gateway 255, 260

Execution sequence 261, 271

Expiration period 603, 610

Expression 272

Expression type 257–259, 269

Extensible Stylesheet Language

 Transformation (XSLT)

mapping 496

External call 303

F

Failed 544, 559

messages 557

Failover 74

faultcode 288

faultstring 288

Field mapping 220

File 139, 159

File Transfer Protocol (FTP) 585, 591

File Transfer Protocol Secure (FTPS) 779

Fingerprint 816, 846

Fingerprints 569

Fire and forget 287

Flexible pipeline 167, 381

Formulas 695

G

Gather 264, 265, 272–275, 279, 280, 301

Gather/merge pattern 264

General splitter 266–268

GET 699

GET by ID 699

Globalization scenarios 153

government formats 154

Granttype 336

Graphical mapper 221

GS1 476

GS1 EANCOM 476

H

Hardware and infrastructure maintenance 534

Hash function 816

Hash value 816

Header 166, 168, 174, 175, 183, 193, 254, 257, 258, 272, 276, 363, 374, 466, 707

Health check 51, 534, 624

High availability 74

failover 74

Hooks 702

Host key 588

HTTP 445

HTTP Access Log 619

HTTP session handling 426, 466

HTTPS 50, 584, 743, 779, 795

Hybrid deployment 33

I

Identity provider 747, 753, 789

IDoc 445, 477, 480

adapter 504, 857

Inbound

authorization 762

Inbound communication 796

Industries 133

Information 697

Innovation

fast 34

InOnly 167, 283, 291, 293

InOut 167, 283, 286

In-progress repository 620

Instance 695, 708, 710, 711, 713, 714, 720, 724, 725, 727

token 714

Integration 33, 35

Integration adapter

artifact 457

Integration Advisor 74, 77, 472, 474, 475

runtime artifacts 475

user roles 478

Integration artifact 68

integration flow 68, 72

Integration bus 36, 37, 44

Integration content 50

artifacts 134

consume 135

deploy 140

design 79

documents 134

export 441

import 441

preconfigured 35

predefined 49, 77

provided by SAP 78

tags 135

terms and conditions 139

transport 441

updates 130

Integration content monitor 184

Integration design

readability 466

Integration developer 62, 126

Integration engine 175, 269, 283, 381

Integration flow 60, 89, 139, 554

change 139

channel 69

component version 421

connectivity details 140

creation 88

definition of 69

demo example 84

deployment 71

design 64, 69

design guidelines 463

development 84

display 139

download 140

endpoint address 101

history 141

model 69

revert to an older version 141

runtime configuration 432

simulation 433

versions 129, 135, 140–145, 148, 153, 186, 423

Integration flow component upgrading 423

Integration flow modeling

palette 92

sender adapter 94

Integration flow step 69

content modifier 93, 412

data store write 97, 99

encryptor 836

exception subprocess 92

local integration process 92

mapping 92

persist 97

router 411

types 92

write variables 97, 403

Integration middleware

on-premise 40

Integration package 50

configure 138

create 157

editor 139

latest 133

modify 138

predefined 40

Integration platform 35, 36, 44, 55, 56

resources 57

virtual 59

virtualized 56

Integration process 164

Integration use case

cloud-to-cloud integration 860

cloud-to-on-premise integration 856

Integration-as-a-service 45

interchange number 509

Interface determination 167

Intermediate certificate 569

Intermediate error 555

International Standards Organization

 (ISO) 477

Internet Message Access Protocol

 (IMAP) 585, 593–595

Internet of Things (IoT) 356

Issuer DN 566, 576

Iterating splitter 267

J

Java Database Connectivity (JDBC) 780

data sources 561, 581, 582

data source 415

Java Database Connectivity (JDBC)

adapter 413

limitations 413

Java Database Connectivity (JDBC) adapter (Cont.)
 parameters 417

Java Message Service (JMS) 305, 307, 780

adapter 307, 605, 621

dead-letter queue 313

expiration period 310

explicit retry 323

monitor 311, 315

queue name 309

receiver 309

resources 611

retry 313

sender 313, 319

transaction handler 320

Java Message Service (JMS) queue 67

JavaScript Object Notation 210

JavaScript Object Notation (JSON) 210

JMS Message Broker 307

JMS queue 307, 600, 605

deletion 312, 609

Join 301

JSON to XML converter 346, 347

K

Key

lifecycle 847

private 774

public 774

X.509 841

Key pair 561, 565, 843

attributes 844

SAP-owned 844

Key renewal 850

Keystore 81, 86, 117, 119, 561, 565, 776, 804, 839, 843

alias 845

managing 843

Keyword 133

Kleopatra 823, 827

Known hosts 588

L

Last modified on 576

LastErrorModelStepId 555

Layout 696

Library of custom type systems 475

Library of type systems 472, 475, 479

Line of business 133

List of entries 263

Load balancer 61, 743, 797

root certificate 797

Local integration process 384, 386, 387, 389

limitations 385

Local process 381, 384, 387

Location ID 795

Locks 139, 620

Log configuration 541

Log content 549

Log details 544

Log files tab 618

Log level 541

trace 66, 369, 468

Logs section 547

M

Mail 593

Mail adapter 341

Mail receiver adapter

parameters 110

Manage integration content 538

Manage locks section 620

Manage security 561, 573, 575, 577, 581

Manage security material 535

Manage stores 311, 600, 605

Manage user and roles 534

Managing tiles 557

Man-in-the-middle attack 774

Mapping 167, 207

editor 217–222

engine 208

step 208

Mapping guidelines (MAGs) 475, 488

editor 488

Mediated communication 37

Mediation engine 165

Memory 534

Mendelson 501

Message content 745

Message disposition notification (MDN) 500

Message end event 394

Message exchange 33, 35, 44, 56

Message exchange pattern (MEP) 167, 266, 283, 285–287, 291, 293

Message flow 164

Message ID 544, 546

Message implementation guidelines

 (MIGs) 475

BUSINESS CONTEXT 484

DIRECTION 484

editor 484

Message implementation guidelines (MIGs)

 (Cont.)

STRUCTURE 485

Message lock 620, 622

Message model 168

Message monitor 184

Message monitoring 293

Message processing 56

Message processing log (MPL) 66, 105, 289, 293, 467, 545, 547, 554, 745

attachments 545, 556, 557

MPL ID 289, 300

properties 554

Message protocol 171

Message queues 82, 605

monitor 311

Message routing 253

Message size 465

Message statuses 559

Message type 220

MessageGuid 555

Message-level security 818, 841

Messaging system 598

Metadata 158

Microservice 74, 743

Microsoft Azure 751

ModelStepId 556

Modifiable 131

Modified by 576

Modularization 381, 397

Monitor 536

Monitoring 51, 79, 80, 104, 117, 119, 121, 127, 179, 184

dashboard 536

data 66, 745

integration content 535

keystore 119

message processing 292, 535, 543

runtime messages 535

Multicast 297, 301

order 302

parallel 297

processing 300

sequential 297, 301

Multimapping 273

Multitenancy 45, 50, 58

N

Namespace 278

Neo 58, 75, 88, 748, 750, 798

Network 743

demilitarized zone 744

sandboxed segment 744

segment 743

New SAP keys 571

Node 555

Notify about update 128–131

Number of concurrent processes 271, 613

Number or queues 612

Number range 82, 509, 614

Number range object 472, 614

O

OAuth 783, 791, 792, 806, 807

access token 790

authorization server 785, 790

client credentials 784

credentials 763, 784, 786, 811

grant type 785, 790

supporting adapters 786

terminology 786

token credentials 784

workflow 785, 791

OAuth2 336

credentials 560

Object Management Group (OMG) 163

OData 192, 349, 445, 477

channel 201

connection 196, 209, 279

service 70, 90, 163, 185, 192–194, 196, 197, 201, 203, 209, 210, 218, 229, 251, 253, 279, 891

OData API

authentication 791

One-way 286, 287

message 167, 283

On-premise 31, 39, 125, 505

OpenAPI 706

OpenPGP 777, 823

Organization 720, 723, 729, 730

Out message 167

Outbound communication 803

Out-of-memory 620

OverallStatus 554

Overlapping routing conditions 260

Overview 141, 697, 698

resources 697

Own content package 157

creating 157

P

Package 168

Parallel gateway 297, 303

Parallel multicast 297

Parallel processing 270

Parameters 129, 142, 144, 186, 701

dynamic 362

Parent process 382–384, 386

Participant 194

Partner directory 472, 518

Passwords 560

PATCH 700

Path 707

Payload 166, 209

Persistence 65

Persona 64

business expert 754

integration developer 754

tenant administrator 754

PGP decryptor 826

PGP encryptor 836

PGP key, user ID 836

PGP public keyring 820, 834

deploy on tenant 834

PGP secret keyring 820, 824

PKCS#7/CMS splitter 268

Platform-as-a-service (PaaS) 56

Point-to-point communication 36

Polling information 345, 540

Pool 164

POST 699

Post Office Protocol version 3 (POP3) 585, 593, 594, 596, 780

Postman 336, 523, 653

Prepackaged integration content 128, 130, 157

import 137

process of consuming 132

update 128, 129

Pretty Good Privacy (PGP)

keyring 565

public keyring 560

secret keyring 560

Principal propagation 793, 794

Process call 384, 387

Process ID 545, 619

ProcessDirect adapter

address 400

limitations 396

use cases 397

Processing 544, 559

chain 165, 276

queue 607

settings 291

Producer 332

Producer integration flow 396

Product 133

Product profile 79, 427, 428

Properties 701

Provider 332

Providers 613

Proxy type 197, 795

Publish content 130

Publisher 332

Publish-subscribe pattern 331, 332

Push-pull pattern 306

PUT 699

Q

Query 707, 733, 737

Query editor 192, 197–201, 207, 209, 218

Queue 333, 334

status 613

Quick configure 128, 129

R

Receiver 129, 140, 144, 186

determination 167

Receiver-initiated message processing 331

Relative path 269

Reliable messaging 304

Request message 220, 222

Request reply 193–196, 202, 203, 207, 210, 222, 266, 279, 280, 286, 348, 357

Request-response message 167

Resource management 465

Resource path 197, 200, 201, 207

Resources 697, 702

Response message 168, 175, 183, 202, 220

Responsibility matrix 534

Restart artifact 540

Restore 570

Result message 176

Retention threshold for alerting 603, 610

Retry 544, 559

messages 557

RFC 780

Robust 287, 291

Robust In-Only 287

Robust One-Way 287

Role 87, 753, 754

creation 764

ESBMessaging.send 87, 95, 755

JSON representation 765, 801

Role collection 742, 752, 754, 755

creation 758

Role template 742

Root cause analysis 293

Root certificate 569

load balancer 799

Route 165, 167, 210

Routing 209

condition 255

rule 260

sequence 302

Run once 358, 361

Runtime configuration 278, 291, 365

Runtime data 66

Runtime node 60

Russian doll 495

S

SAP API Business Hub 77

SAP Ariba 146

SAP Business Process Management

 (SAP BPM) 164

SAP Cloud for Customer 50, 146, 148, 149, 855

content 148

SOAP adapter 148

SAP Cloud Platform 56

account 58

cockpit 73, 76

environment 58

regions 748

service instance 87

service key 87, 103

tenant 58, 63

transport management service 442, 443

trial 841

trial account 86

user management 753

SAP Cloud Platform cockpit 742

SAP Cloud Platform Connectivity 42, 49, 171, 197, 505, 513, 585, 599

SAP Cloud Platform Enterprise

Messaging 334, 336–338, 340–344, 349, 351, 352

REST APIs 336

test 353

SAP Cloud Platform environment

Neo 798

SAP Cloud Platform Integration 33, 55, 125, 533, 624, 625, 693

API 65

architecture 72, 741

connectivity options 147

day-to-day operations 534

monitoring 535

monitoring capabilities 533

operational tasks 534

operations 533

regions 751

releases updates 535

runtime address 102

update cycle 428

SAP Cloud Platform Integration

Suite 471, 693

SAP Cloud Platform Open Connectors 694

SAP data center 50, 748

SAP ERP 855

SAP ERP Human Capital Management

 (SAP ERP HCM) 146

SAP ERP Material Management

 (SAP MM) 152

SAP Events 340

SAP Exchange Infrastructure (XI)

adapter 600, 606, 621

SAP Fiori 860

SAP ID service 753

SAP Integration Content Catalog 40, 50, 125–130, 132–134, 137, 146–148, 153, 154, 157, 160, 625, 691, 693

accessing 126

catalog 126

content 126

discover 128

existing package 133

package 126

prebuilt integration flows 126

search 132

templates 126

via a publicly accessible URL 126

via your own tenant 126

web-based application 127

SAP Key History 571

SAP Keys 570

SAP Process Integration 40, 164, 167, 208–210, 220, 221, 223, 264

SAP Process Orchestration 40, 79, 426, 427, 534

update cycle 427

SAP S/4HANA	860
SAP S/4HANA Cloud Enterprise Edition	860
SAP Solution Manager	164
SAP SuccessFactors	50, 125, 146, 147, 862
<i>adapter</i>	147
<i>content</i>	146
<i>discover all packages</i>	147
<i>Employee Central</i>	862
<i>integrates onboarding</i>	146
<i>OData API</i>	863
<i>query, insert, upsert, update</i>	866
<i>SFAPI</i>	863
SAP SuccessFactors adapter	864
<i>query operations</i>	865
SAP Supplier Relationship Management (SAP SRM)	152
SAP_ApplicationID	544
SAPJMSRetries	324
SASL	343
Scaling, horizontal	45, 51
Schedule on day	358, 359
Schedule to recur	359, 360
Scheduler	144, 359
Script	468
Search	696
Secret key	830
Secure communication, HTTPS	68
Secure parameter	560, 811
Secure Shell (SSH)	50, 584, 588
<i>key</i>	839
<i>known hosts</i>	560, 565
Secure Shell File Transfer Protocol (SFTP)	164, 445, 584, 588
Security	45, 68, 695
<i>data storage security</i>	745
<i>message level</i>	772
<i>message-level</i>	815
<i>physical</i>	748
<i>software development process</i>	749
<i>transport-level</i>	772, 779
Security Material	561, 562
Security standard	
<i>OpenPGP</i>	817
<i>PKCS#7</i>	817
<i>S/MIME</i>	818
<i>WS-Security</i>	818
<i>XML Advanced Electronic Signatures</i> (<i>XAdES</i>)	818
<i>XML Signature</i>	817
Security standards	817
Send	303
Sender	129, 140, 143, 144, 186
Separation of concerns	381
Sequence flow	177
Sequential Multicast	297, 301, 349
Serial number	576
Service definition	266, 285, 290
<i>Web Services Description Language</i> (<i>WSDL</i>)	290
Service instance	763, 800
Service key	763, 788, 789, 800
Service plan	790
Settings	127
Setup	697
Signature	773
Signing	500
Simple Expression Language	111, 176, 200, 258
Simple Mail Transfer Protocol (SMTP)	585, 593
Simple Object Access Protocol (SOAP)	164, 171, 179, 192, 193, 222, 254, 260, 275, 282, 283, 285, 287, 291–293, 304, 445, 477
<i>channel</i>	254, 285, 287, 290, 291, 293, 304
<i>fault</i>	288
<i>processing settings</i>	287
<i>Web Services Description Language</i> (<i>WSDL</i>)	290
Simple Object Access Protocol (SOAP) adapter	868
Simulation	433
<i>end point</i>	437
<i>mode</i>	435
<i>response</i>	436
<i>start point</i>	435
SOAP adapter	171, 172
<i>address</i>	95
Software	
<i>maintenance</i>	45
<i>update</i>	51, 68
Software updates and maintenance	534
Software-as-a-service (SaaS)	34
<i>administrator</i>	751
Splitter ... 264–268, 270, 272–274, 276, 279, 280	
<i>pattern</i>	264
<i>validation error document</i>	515
Splitting tag	268, 269
SQL	418
SRTIDOC	507
Start event	164, 171
Start message event	265
Start timer event	357
StartTime	554

Status	544, 545, 555, 558
<i>details</i>	540
StepID	556
Stop on exception	271
StopTime	554
Stored	563
Streaming	270, 468
Structuring large integration flows	381
Subject DN	566, 576
Subprocess	381–384, 387, 468
Subscription	333
Supported platform	133
Swagger	704
Symmetric key technology	773
Synchronous communication	192
Synchronous interface	219, 220
Synchronous message handling	167, 281
System	720
System Log Files tile	618
Tags	135
Talent management process	146
TCP	342, 780
Templated resources	719
Templates, copy	137
Temporary data	600
Tenant	127, 132, 139, 140, 142, 144
<i>database</i>	63
<i>tenant isolation</i>	58
Tenant administrator	111, 116, 754
Tenant isolation	740, 742
<i>message processing</i>	742
<i>user management</i>	742
Terms and condition	139
Test the API Docs	708
Tile settings	558, 559
Tiles	536, 543, 557, 558
Time to live (TTL)	351
Time-based integration flow	357
Timer	356
Timer start event	359–362, 384
Timer-based message handling	355
Timer-based message transfer	356
tokenendpoint	336
Topic	332, 335
Trace	542
Transaction handling	320, 321, 465
Transactions	614
Transport	
<i>destination</i>	443
Transport (Cont.)	
<i>landscape</i>	442
<i>manual</i>	441
Transport Layer Security (TLS)	343, 584, 586, 840
Transport node	445
Transport protocol	779
<i>HTTPS</i>	61
Transport-level security	840
Traversal path	548
Troubleshooting	184
Trust configuration	757
Try it out	715, 732
Twitter adapter	806, 812
Twitter API	809
Type of connectors	696
U	
UN/CEFACT	476
UN/EDIFACT	476
Undeploy	540
Update available	130
Update package	130
Upgrade	34
URI	192
URL	139, 159, 192, 193, 202
User administration	752
User management	752, 753
User management, dialog user	742
User role	82, 266, 561, 573
Username	560, 576
User-to-role assignment	742
V	
Valid until	576
Validate server certificate	586, 592
Validation	697, 703
Value mapping	90, 139, 159
Variable	66, 82
<i>global</i>	403
<i>local</i>	403
<i>monitoring</i>	406
Variables	603
Variables tile	604
Version history	141
Versioning	420
Virtual machine	
<i>Java Virtual Machine (JVM) instance</i>	60
Visibility	601, 602, 604

W

Web of trust 777

Web service 166

Web Services Description Language

 (WSDL) 174, 209, 210, 217, 223, 264, 285, 286, 290, 291

Web UI 64, 78, 381, 446

address 87

Monitoring 80

WebShop

application 356

WebSocket 342

Worker 58, 63, 102, 744, 761

failure 74

Working with lists 262

Write Variables 604

WS-Addressing 266

WS-Standard 287, 289, 291

X

X.509 776, 839

XML 201, 209, 210, 257–259, 262, 264, 273, 278, 291

envelope 273

schema definition 209

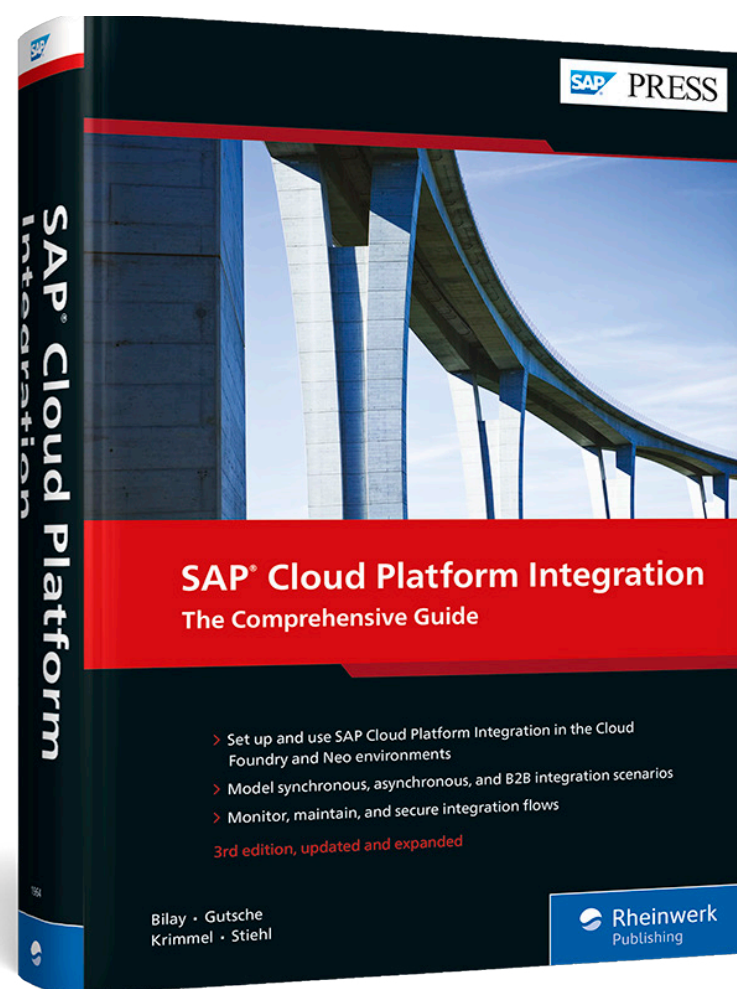
validator 292, 495, 496

XML Path Language (XPath) 175

XML Path Language (XPath) expression 257, 258, 269, 273, 274, 276, 278, 279

XML Schema Definition (XSD) 201, 209–211, 217, 218, 223

XML to EDI Converter 473



John Mutumba Bilay, Peter Gutsche, Mandy Krimmel, Volker Stiehl

SAP Cloud Platform Integration: The Comprehensive Guide

906 Pages, 2020, \$89.95

ISBN 978-1-4932-1964-3



www.sap-press.com/5077



John Mutumba Bilay studied computer engineering and finance at the University of Cape Town, South Africa. After completing his studies, he started his career as a software engineer. He currently works as a senior software engineer and enterprise integration consultant at Rojo Consultancy B.V. in the Netherlands.



Dr. Peter Gutsche studied physics at Heidelberg University, Ruperto Carola. After completing his Ph.D, he joined SAP in 1999. As a technical author, Peter was involved in many knowledge management projects related to SAP's interface and integration technologies.



Mandy Krimmel studied engineering at Humboldt University, Berlin, Germany. In 1998, she started her professional career at a research institute of the German state of Baden-Württemberg, where she was responsible for the technical evaluation of various EU research projects. In 2001, Mandy joined SAP, working on various integration-related projects, including SAP Process Integration and SAP Cloud Platform Integration.



Prof. Dr. Volker Stiehl studied computer science at the Friedrich-Alexander-University of Erlangen-Nuremberg. After 12 years as a developer and consultant at Siemens, he joined SAP in 2004. As chief product expert, Volker was responsible for the success of the products SAP Process Orchestration, SAP Process Integration, and SAP HANA Cloud Integration.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.