SAP API Management
# Content

## 1  SAP API Management in the Cloud Foundry Environment

1.1 What is API Management ....................................................... 5  
   Components of API Management ........................................ 6  
   Concepts of API Management ........................................... 9  
   API Services ..................................................................... 10  

1.2 What's New for SAP API Management .......................... 11  
   Archive - Release Notes for SAP API Management .......... 16  

1.3 Patch Releases for API Management .............................. 40  

1.4 Configure . ................................................................. 42  
   Initial Setup ..................................................................... 42  
   Custom Domain Configuration for API Portal or API Business Hub Enterprise Subscription .... 69  
   Requesting for Two-Way SSL Certificate .................. 71  
   Requesting Egress NAT IP Addresses for API Management Service in Different Regions .... 71  
   Creation of Shadow Users ........................................ 72  
   Cancelling API Management Service Subscription .... 72  
   Centralized API business hub enterprise ..................... 73  

1.5 Development ............................................................. 79  
   Build APIs ..................................................................... 80  
   Publish APIs ................................................................... 481  
   Analyze APIs .................................................................. 488  
   Consume APIs ................................................................ 503  
   Monetize APIs ................................................................ 520  
   Discover Packages ...................................................... 533  
   Transport APIs and Its Related Artifacts .................. 534  

1.6 API Documentation ......................................................... 562  
   API Portal - API Proxy (CF) ............................................. 562  

1.7 Security ................................................................. 576  
   Data Protection and Privacy ........................................... 577  

1.8 Monitoring and Troubleshooting .................................. 620  
   Limits ........................................................................... 620  
   Monitor the Health of Custom Domain Virtual Host Certificates Using SAP Cloud ALM .... 623  

## 2  Migration Assistance for API Management from Neo to Cloud Foundry Environment

2.1 Prerequisites ............................................................... 624  

2.2 Clone API Management Artifacts .................................. 626  
  Cloned and Uncloned Entities ......................................... 642
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tenant Cloning Tool Behavior</td>
<td>643</td>
</tr>
<tr>
<td>2.3 Post Cloning Tasks</td>
<td>644</td>
</tr>
<tr>
<td>2.4 Recommendations</td>
<td>648</td>
</tr>
<tr>
<td>2.5 Security</td>
<td>649</td>
</tr>
<tr>
<td>3 Migrating API Management Subscription</td>
<td>650</td>
</tr>
<tr>
<td>Created Using the Starter Plan Service Instance</td>
<td></td>
</tr>
<tr>
<td>4 Migrating API Management Subscription</td>
<td>652</td>
</tr>
<tr>
<td>Created Using the Starter Plan Service Instance</td>
<td></td>
</tr>
<tr>
<td>to Different Subaccounts</td>
<td></td>
</tr>
</tbody>
</table>
SAP API Management in the Cloud Foundry Environment

SAP API Management lets you publish, promote, and oversee APIs in a secure and scalable environment.

**Environment**

This service runs in the following environments:

- NEO environment
- Cloud Foundry environment

**Features**

- **Create omni-channel experiences**
  - Use API Designer and Open APIs to create a omni-channel mobile experience across devices.

- **Secure your digital assets, interfaces**
  - Help protect your data and digital assets in this hyper-connected world. Get deep insights on API usage.

- **Manage the end-to-end lifecycle of APIs**
  - Scale billions of API calls to unlock new opportunities, new business potential and add additional value.

- **Engage developers and partners**
  - API Business Hub Enterprise simplifies sharing managed APIs and collaborations with customers, partners, and developers.

- **Grow new revenue streams**
  - Monetize your data and digital assets with help of API Portal. Upsell and cross-sell through your ecosystem.

- **Evolve B2B integrations**
  - Extend solutions with additional SAP BTP capabilities for mobile, offline and integration.

API Management technology helps you to share digital assets and enable developer communities to consume these assets in new channels, devices, and user interfaces. Available in the cloud, the technology helps promote coinnovation among employees, partners, and the developer community. To gain better insights about consumer needs, you can empower employees and partners with access to critical information and increase reach to a wider customer base.

The Cloud Foundry environment gives you the ability to subscribe to the API Management service, while you may choose a public infrastructure to run the API Management service, such as Amazon Web Services or microsoft Azure.
Get started by subscribing to API portal and API business hub enterprise applications where you can create APIs and consume them. For setting up the API Portal application, see here [page 43]. Once the API portal is setup, see here [page 49] to set up the API business hub enterprise application.

1.1 What is API Management

Create simple digital experiences for your consumers, partners, and employees.

SAP API Management lets you publish, promote, and oversee APIs in a secure and scalable environment.

The Cloud Foundry environment gives you the ability to subscribe to the API Management service, while you may choose a public infrastructure to run the API Management service, such as Amazon Web Services or Microsoft Azure.

→ Tip

This is the documentation for API Management for Cloud Foundry. If you are looking for information about the Neo environment, see here.

Features

Create omni-channel experiences

Use API Designer and Open APIs to create an omni-channel mobile experience across devices.

Secure your digital assets, interfaces

Help protect your data and digital assets in this hyper-connected world. Get deep insights on API usage.

Manage the end-to-end lifecycle of APIs

Scale billions of API calls to unlock new opportunities, new business potential and add additional value.

Engage developers and partners

API Business Hub Enterprise simplifies sharing managed APIs and collaborations with customers, partners, and developers.

Grow new revenue streams

Monetize your data and digital assets with help of API Portal. Upsell and cross-sell through your ecosystem.

Evolve B2B integrations

Extend solutions with additional SAP BTP capabilities for mobile, offline and integration.

Use Cases

With the emergence of cloud, mobile and social technologies, new applications have become a driving force in the way people consume content and access services. Millions of mobile devices in use today are generating
digital data at an exponential rate. This massive influx of digital information is changing the way businesses are operating. To keep up with the digital footprint produced, businesses identify and implement ways to reach out to their customer and meet their needs, easily and securely.

Through software interfaces called application programming interfaces (APIs), companies can provide business services and information directly to customers. APIs simplify the work of programming graphical user interface components for all types of apps on mobile devices, in the cloud, and on wearables. Exposing digital assets enable you to create and deliver content and business services to your customers, partners, and employees. That way they can better engage, collaborate, and innovate.

API Management technology helps you to share digital assets and enable developer communities to consume these assets in new channels, devices, and user interfaces. Available in the cloud, the technology helps promote coinnovation among employees, partners, and the developer community. To gain better insights about consumer needs, you can empower employees and partners with access to critical information and increase reach to a wider customer base.

API Management facilitates consumer engagement anywhere, any time. It reduces complexity by leveraging a single provisioning platform (API Platform) to provide unified access and governance of APIs across a heterogeneous landscape.

Get started by subscribing to API portal and API business hub enterprise applications where you can create APIs and consume them. For setting up the API Portal application, see here [page 43]. Once the API portal is setup, see here [page 49] to set up the API business hub enterprise application.

### 1.1.1 Components of API Management

API Management component overview

The API Management infrastructure consists of five components: API Runtime, API Portal, API business hub enterprise, API Analytics, and API Designer.
API Management

API Runtime

API Portal

API Business Hub

Enterprise

API Analytics

API Designer

- #unique_12/unique_12_Connect_42_subsection-im1 [page 8]
- #unique_12/unique_12_Connect_42_subsection-im2 [page 8]
- #unique_12/unique_12_Connect_42_subsection-im3 [page 8]
- #unique_12/unique_12_Connect_42_subsection-im4 [page 8]
- #unique_12/unique_12_Connect_42_subsection-im5 [page 8]

Hover over each element for a description. Click the element for more information.
**API Runtime**
Allows you to deploy and productively use your APIs. Applications consume the API Runtime, request for API authentication, and access.

**API Portal**
The one-stop-shop to create, secure, and publish API Proxies. This is the place for easy discovery of APIs, and you the API Administrator, can manage, meter, secure your APIs, as well as define and publish rate plans. To know more about using the API Portal see, Development [page 79].

**API Business Hub Enterprise**
Self-service for application developers to discover, browse, and explore APIs, subscribe to rate plans, and build apps. To know more about using the API business hub enterprise, see Consume APIs [page 503].

**API Analytics**
Provides powerful analytical tools to track your API usage. Use API Analytics to collect information on the URL, user ID for API call information, latency data, and so on. To know more about API Analytics, and how you can use it, see Analyze APIs [page 488].

**API Designer**
API developers can define, implement, and document APIs. It provides open API support, and a variety of outputs can be generated. For more information on the API Designer, see, here.

You can use API Management with one of SAP’s numerous in-house API providers as well as any non-SAP APIs. API Management leverages your investment in SAP solutions and can also be integrated with non-SAP solutions. It helps unlock the value of digital assets and enables you to create and deliver content. API Management enables applications to run seamlessly by accessing backend data securely. It provides one-experience for managing and monitoring APIs across various data platforms with real-time Analytics.

The following image shows how different stakeholders interact with the various API Management components, and how API Management in turn interacts with the different cloud and on-premise systems.
1.1.2 Concepts of API Management

Structure of the API Portal for API Management

Before you start to build APIs, it is important to understand the structure of the API Portal. Its structure defines how APIs, products, applications, users, developers, and accounts are all related to each other within API Management.

The structure of the API Portal comprises:

- API Management Account
- System
- User
- API
- Product
- Developer
- Application
- App Key

The table describes the various entities that comprise the API Portal:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management Account</td>
<td>An API Management account is the highest level of data hierarchy. An account is a representation of all components including APIs, products, applications, systems, users, and developers.</td>
</tr>
<tr>
<td>System</td>
<td>In API Management, <strong>System</strong> refers to the API provider systems where the actual backend services reside. System could either be an ABAP system, SAP Gateway system, Enterprise Services Repository, or systems that host generic REST services or third party provider systems. API Management allows you to add and manage an API provider system. After you have added a system, you can browse for the APIs in that system.</td>
</tr>
<tr>
<td>User</td>
<td>API Management can have multiple users. Different users have different roles and privileges assigned. For example, people who create APIs and products or analyze the metrics or the application consumer who can access the APIs provisioned by API Management.</td>
</tr>
</tbody>
</table>
APIs are Application Programming Interfaces. They comprise a set of routines, protocols, and tools for building software applications. APIs define sets of requirements that govern how applications communicate with one another. They facilitate interaction by selectively exposing certain functionalities, allowing different applications, websites, or devices to communicate effectively with each other.

**i Note**
API Management supports OData, REST, and SOAP services.

A product is a bundle of APIs. It contains metadata specific to your business for monitoring or analytics. For example, all APIs related to CRM can be bundled as one CRM product. API Management collects data for analyzing the products.

One or more developers can create applications in the API Management account. A developer can consume the APIs, but cannot create APIs.

To create an application, the developer must have registered the account. After having created an application, the developer uses the app (application) key to consume the APIs.

Applications include the Web or mobile applications that consume the exposed APIs. When you create an application, you select the product to include in this application. For each application that you create, API Management generates an app key and secret. Use this key to gain access to multiple products. Developers create one or more applications using the APIs you expose.

Based on the authorization mechanism you define for your APIs, the application passes an app (application) key together with every request to your APIs. If that key is valid, the request is permitted. API Management supports different types of authentication, such as a simple API key, OAuth, and so on.

### 1.1.3 API Services

From API Management, a variety of APIs are offered as services in specific use cases and workflows. You can explore them and try it out in the SAP API Business Hub in the following url: [http://api.sap.com](http://api.sap.com).

<table>
<thead>
<tr>
<th>Services</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Portal</td>
<td>You can browse through this API package for API Admin services with the required resources.</td>
</tr>
<tr>
<td>API business hub enterprise</td>
<td>You can browse through this API package for application developer services that are offered.</td>
</tr>
</tbody>
</table>
## 1.2 What's New for SAP API Management
<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Action</th>
<th>Lifecycle</th>
<th>Typ</th>
<th>Line of Business</th>
<th>Avail. as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>Cloud Foundry</td>
<td>Custom Domain Configuration</td>
<td>The custom domain feature is now enabled for API Management as a capability within the Integration Suite product. For more information, see Custom Domain Configuration for API Portal or API Business Hub Enterprise Subscription [page 69].</td>
<td>Info only</td>
<td>General</td>
<td>New</td>
<td>Service</td>
<td>202-03-11</td>
</tr>
</tbody>
</table>

Custom domain feature is now enabled for API Management as a capability within the Integration Suite product. For more information, see Custom Domain Configuration for API Portal or API Business Hub Enterprise Subscription [page 69].
### Technical Component

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
<th>Action</th>
<th>Lifecycle</th>
<th>Type</th>
<th>Line of Business</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skipping the cloning of Application Key and Secret in side by side migration</td>
<td>If you want to skip the cloning of Application Key and Secret in side by side migration, then set the &quot;skipApplicationKeySecretCloning&quot; flag to true. For more information, see the Procedure section in Clone API Management Artifacts [page 626]. Also, a new version of the tenant cloning tool</td>
<td>Info only</td>
<td>General Availability</td>
<td>New Service</td>
<td>Not applicable</td>
<td>2022-02-15</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Environment</td>
<td>Title</td>
<td>Description</td>
<td>Action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.6.2 is now available. For more information, see Clone API Management Artifacts [page 626].</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical Component</td>
<td>Environment</td>
<td>Title</td>
<td>Description</td>
<td>Action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>API Management</td>
<td>Cloud Foundry</td>
<td>Migrating API Management Subscription Created Using the Starter Plan Service Instance to a Different Sub-account</td>
<td>If you have Integration Suite Premium edition license available in a different sub-account, then API Management design time subscription can be migrated to this different sub-account as well. For more information, see Migrating API Management Subscription Created Using the Starter Plan</td>
<td>Info only</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lifecycle</th>
<th>Typ e</th>
<th>Lin e of Bus in ness</th>
<th>Ava ilable as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>New</td>
<td>Service</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Availabil ity</td>
<td></td>
<td></td>
<td>202</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>115</td>
</tr>
</tbody>
</table>
1.2.1 Archive - Release Notes for SAP API Management

Archive of SAP API Management release notes.

- 2021 Archives [page 17]
- 2020 Archives [page 24]
- 2019 Archives [page 29]
- 2018 Archives [page 33]
### 1.2.1.1 2021 Archives

<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Capability</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Action</th>
<th>Type</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Monitor the Health of Custom Domain Virtual Host Certificates</td>
<td>You can use SAP Cloud Application Lifecycle Management (ALM) application for monitoring the health of API Management certificates. For more information, see <a href="#">Monitor the Health of Custom Domain Virtual Host Certificates Using SAP Cloud ALM</a>.</td>
<td>Info only</td>
<td>New</td>
<td>2021-12-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update to the Client SDK</td>
<td>You can now apply policy template to an existing API proxy. For more information, refer to the Client SDK version 1.4.0 in <a href="#">API Services</a>.</td>
<td>Info only</td>
<td>New</td>
<td>2021-09-29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Applications table on My Workspace</td>
<td>The My Workspace section in API Business Hub Enterprise has been revamped to provide better performance. The Cost Incurred column has been removed from the Applications table. You can view the cost incurred details in the Cost section. For more information, see <a href="#">Create an Application</a>.</td>
<td>Info only</td>
<td>New</td>
<td>2021-09-29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SAP Analytics Cloud for API Management</td>
<td>You can use the API Management Reporting Dashboard on SAP Analytics Cloud to monitor API usage and performance through various API metrics and KPIs. For more information, see <a href="#">SAP Analytics Cloud for API Management</a>.</td>
<td>Info only</td>
<td>New</td>
<td>2021-08-31</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Capability</td>
<td>Environ-ment</td>
<td>Title</td>
<td>Description</td>
<td>Action</td>
<td>Type</td>
<td>Date</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>----------------------</td>
<td>--------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Overriding the default update operation for API Proxy of Type ODATA</td>
<td>When discovering an API from the on-premise SAP Gateway system via OData API Provider, for the entities that are defined as “sap:updatable” in the backend service, you can choose the update operation (&quot;PUT&quot; and &quot;PATCH&quot;) for OData V2 and OData V4 respectively. For more information, see Overriding the Default Update Operation for API Proxy of Type OData [page 403].</td>
<td>Info only</td>
<td>New</td>
<td>2021-08-31</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Product Transport</td>
<td>When transport is triggered for a Product, all the entities of the Product get transported along with the Product. For more information, see Transporting a Product from Source to Destination [page 559].</td>
<td>Recommended</td>
<td>New</td>
<td>2021-08-09</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Designer</td>
<td>For a given Resource, you can use the API Designer to detect and correct the errors in swagger definition, and save the changes. For more information, see the note in step 14 in Create an API [page 393].</td>
<td>Info only</td>
<td>New</td>
<td>2021-08-09</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Capability</td>
<td>Environment</td>
<td>Title</td>
<td>Description</td>
<td>Action</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Save and Deploy API Proxy</td>
<td>Saving is a design time activity; at this stage, multiple aspects of the proxy might change. Until all the changes made to the proxy are considered and are finally saved, the proxy should not be deployed.</td>
<td>Required</td>
<td>New 2021-08-09</td>
<td></td>
</tr>
</tbody>
</table>

**Action:** Previously, editing and then saving the changes in an already deployed API, would deploy the changes in runtime. Now, after saving the changes you’ve made to the API proxy, you have to choose **Deploy** for the latest changes to reflect in runtime. For more information, see **Edit an API Proxy** [page 459].

**Notes:**

- API proxies always get imported to the destination API portal in the deployed state. For more information, see **Transporting an API Proxy from Source to Destination** [page 554]. Additionally, APIs attached to the Product get imported to the destination API portal in the deployed state. For more information, see **Transporting a Product from Source to Destination** [page 559].
- When an API proxy is transported or exported individually or as a part of a Product, by default, it gets imported to the target in the deployed state. For more information, see **Import an API** [page 448].
- If you try to publish a Product that has an API with saved changes attached to it, you get a warning message that there are changes in the APIs that aren’t deployed yet. Similarly, you’ll receive a warning message if you try to publish a Product which has multiple APIs attached to it, and few of these APIs have changes that are saved but not deployed. For more information, refer the note in step 9 in **Create a Product** [page 482].
<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Capability</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Action</th>
<th>Type</th>
<th>Available as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Auditing and Logging Information</td>
<td>Here you can find a list of the security events that are logged by TECHNICAL COMPONENT. For more information, see Auditing and Logging Information [page 581].</td>
<td>Info only</td>
<td>New</td>
<td>2021-08-09</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Business Hub Enterprise</td>
<td>You can now update the credentials used to establish a connection between the API portal and the API Business Hub Enterprise for a submitted request and an approved request. Action: To update the credentials, see Updating the Connection Request Credentials [page 76] and Updating the Connection Request Credentials for an Approved Request [page 78].</td>
<td>Recommended</td>
<td>New</td>
<td>2021-07-09</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Migrating API Management Subscription Created Using the Starter Plan Service Instance</td>
<td>You can now migrate the design-time components that you have in the Neo environment, which was previously set up using Starter Plan instance, to the Cloud Foundry environment, keeping the runtime components as is. Action: To migrate the design-time components from Neo to Cloud Foundry, see Migrating API Management Subscription Created Using the Starter Plan Service Instance [page 650]</td>
<td>Recommended</td>
<td>New</td>
<td>2021-07-06</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Custom Domain Configuration for API Portal or API Business Hub Enterprise Subscription</td>
<td>To complete the process of configuring a custom domain for the API Portal or the API Business Hub Enterprise application using the Custom Domain Service in the SAP BTP Cloud Foundry environment, you need to contact the SAP API Management operations team. For more information, see Custom Domain Configuration for API Portal or API Business Hub Enterprise Subscription [page 69].</td>
<td>Recommended</td>
<td>New</td>
<td>2021-06-25</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Capability Environment</td>
<td>Title</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------</td>
<td>-------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| API Management      | Integration Suite      | Update to the Client SDK | You can now export policy templates from the API portal.  
Action: For more information on how to export the policy templates, refer to the Client SDK version 1.3.0 in API Services [page 10]. |
| API Management      | Integration Suite      | Generate client ID and secret from API Portal UI | You can now generate the credentials from the API Portal to establish the connection with centralized API Business Hub Enterprise.  
Action: See the Next Steps section in Setting Up API Portal Application [page 43]. See the Prerequisites section and the Note in the Results section in Creating a Connection Request for the Centralized API business hub enterprise [page 74]. |
| API Management      | Integration Suite      | Configuring Load Balancing for API Proxy from API Portal | You can now configure the load balancer to distribute the load efficiently across multiple API providers. For more information, see Configuring Load Balancing [page 480]. |
| API Management      | Integration Suite      | Region specific NAT IP addresses | To get region specific NAT IP addresses (egress, IPs for request from API Management), raise a support ticket. For more information, see Requesting Egress NAT IP Addresses for API Management Service in Different Regions [page 71]. |
| API Management      | Integration Suite      | Converting Externally Managed APIs to Internally Managed APIs | You can convert an external API, whose lifecycle isn’t be managed by SAP API Management to an internal API so that management capabilities can be enabled for that API.  
Action: To convert an external API to an internal API, see Converting Externally Managed APIs to Internally Managed APIs [page 83]. |
<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Capability</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Action</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Updating the Connection Request Credentials</td>
<td>There can be instances where you have to update the credentials that you’ve used to establish a connection between the API portal and the API Business Hub Enterprise. For more information, see Updating the Connection Request Credentials [page 76].</td>
<td>Recommended</td>
<td>New 2021-04-12</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Perform Additional Tasks in API Designer</td>
<td>To download the API swagger specifications, choose Download and select JSON or YAML format. For more information, see Perform Additional Tasks in API Designer [page 446].</td>
<td>Info only</td>
<td>New 2021-04-12</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Transporting API Providers, Certificates, and Key Value Maps</td>
<td>You can now trigger the transport of API Provider, Key Store Certificate, Trust Store, and Key Value Maps individually. Action: To trigger the transport of API Provider, Key Store Certificate, Trust Store, and Key Value Maps individually, see Triggering Content Transport Using SAP Cloud Transport Management Service [page 554].</td>
<td>Recommended</td>
<td>New 2021-04-12</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Update to the Client SDK</td>
<td>Two new commands to create API Proxies either by providing parameter information or by uploading JSON files have been added. For more information, see API Services [page 10].</td>
<td>Info only</td>
<td>New 2021-03-16</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Content Transport Using Cloud Transport Management Service</td>
<td>You can now use the SAP Cloud Transport Management service for exporting, importing, and shipping the API Management content from the Development or Test environment to Production environment. Action: To transport API Management content from the Development or Test environment to Production environment, see Transport APIs and Its Related Artifacts [page 534].</td>
<td>Recommended</td>
<td>New 2021-03-16</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Capability</td>
<td>Environment</td>
<td>Title</td>
<td>Description</td>
<td>Action</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Configuring Health Monitor and Load Balancer for API Proxy using .Zip</td>
<td>Configure health monitor and the load balancer to put the active target server in rotation and to distribute the load efficiently across multiple servers. For more information, see <a href="#">Load Balancing Across API Providers</a>.</td>
<td>Recommended</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>UI changes in the SAP BTP Cockpit</td>
<td>A new path has been added on the UI to subscribe to the application plan for API portal and API Business Hub Enterprise. For more information, refer to the step 3 in the following topics: <a href="#">Setting Up API Portal Application</a> <a href="#">Setting Up API business hub enterprise Application</a>.</td>
<td>Info only</td>
<td>Change</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Business Hub Enterprise</td>
<td>You can now maintain a centralized API catalog in one API Business Hub Enterprise that accepts contents like API proxies, API products, and so on, from multiple API portals. Action: You can select multiple products published from the same portal but you can’t select products published from different portals. For more information, see <a href="#">Centralized API business hub enterprise</a> and <a href="#">Create an Application</a>.</td>
<td>Recommended</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Discover REST and SOAP APIs</td>
<td>Discover REST and SOAP APIs along with OData APIs while creating a proxy for Integration flow. For more information, see <a href="#">Creating an API from SAP Cloud Integration API Provider</a>.</td>
<td>Info only</td>
<td>New</td>
</tr>
</tbody>
</table>

**Parent topic:** [Archive - Release Notes for SAP API Management](#)

## Related Information

- [2020 Archives](#)
- [2019 Archives](#)
- [2018 Archives](#)
### 1.2.1.2 2020 Archives

<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Capability</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Type</th>
<th>Available as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Edit APIs with an in built API designer</td>
<td>You can edit your APIs using the API designer, which is now embedded in the API Portal. For more information, see Edit an API Proxy [page 459].</td>
<td>New</td>
<td>202-12-23</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>List externally managed APIs on the API Portal</td>
<td>You can now import and list externally managed APIs on the API Portal. For more information, see Externally Managed APIs [page 82]</td>
<td>New</td>
<td>202-12-02</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Consume Integration flows more securely with OAuth Client credentials support for CPI Providers.</td>
<td>You can now use OAuth2ClientCredentials when creating an API provider. For more information see, Create an API Provider [page 374]</td>
<td>New</td>
<td>202-12-02</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API state can be entered during import, and is available during export of an API.</td>
<td>For more information on the details of the API state to be provided during import, see Import an API [page 448].</td>
<td>New</td>
<td>202-12-02</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Capability</td>
<td>Environment</td>
<td>Title</td>
<td>Description</td>
<td>Type</td>
<td>Available as of</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| API Management      | Integration Suite | Cloud Foundry | API Runtime | The update in the API Runtime has caused the following changes:  
- In the JWT policy, validation may fail if the RSA keys are smaller than 2048 bits.  
- The Concurrent Rate Limit Policy has been deprecated.  
- In the ExtractVariables policy when an XML variable is not resolved via an XPath expression, an error occurs. So, continueOnError should be set to true or IgnoreUnresolvedVariables set to true to allow execution of the policy. | Changed | 202-10-27 |
<p>| API Management      | Integration Suite | Cloud Foundry | Client SDK | The client SDK is now available, for more information see [API Services](page 10). | New | 202-10-27 |
| API Management      | Integration Suite | Cloud Foundry | Advanced API Analytics | Advanced API Analytics brings to you the all new analytics dashboard, providing handy and powerful analytical reporting tools to track your API performance and usage. For more information, see [Advanced API Analytics](page 492). | New | 202-10-27 |
| API Management      | Integration Suite | Cloud Foundry | Shadow user creation | There is a new process for shadow user creation, for more information, see [Creation of Shadow Users](page 72). | Changed | 202-10-21 |
| API Management      | Integration Suite | Cloud Foundry | Versioning | You can now version your APIs. For more information, see [API Versioning](page 391). | New | 202-10-08 |
| API Management      | Integration Suite | Cloud Foundry | Migration from Neo environment to Cloud Foundry | You can now choose to clone the API Portal and API business hub enterprise entities at different times during migration. For more information, see [Clone API Management Artifacts](page 626). | New | 202-10-08 |
| API Management      | Integration Suite | Cloud Foundry | Embedded API Designer | The API Designer is now embedded within the API Portal, allowing you to create and update your APIs in the same space. You will find some changes in the API designer, such as a shift in the editor to the right side of the screen, and a change in the theme, moving to a brighter background to align with the API portal. | Changed | 202-10-08 |</p>
<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Capability</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Type</th>
<th>Available as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Create an API for an Integration Flow</td>
<td>Users can discover Integration Flows through the Cloud Integration API Provider and generate APIs for the same. For more information, see Creating an API from SAP Cloud Integration API Provider [page 401].</td>
<td>New</td>
<td>2020-09-09</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Create an API Provider of type Cloud Integration</td>
<td>Users can now create an API Provider of type “Cloud Integration” to connect to a Cloud Integration system, discover Integration Flows through the API Provider and generate APIs for the same. For more information on creating an API Provider of type Cloud Integration, see Create an API Provider [page 374].</td>
<td>New</td>
<td>2020-09-09</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Services</td>
<td>From API Management, a variety of APIs are offered as services in specific use cases and workflows. For more information, see API Services [page 10]</td>
<td>New</td>
<td>2020-08-11</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Request for an Additional Virtual Host in Cloud Foundry Environment</td>
<td>Create a new virtual host or update an alias for an existing virtual host in the Cloud Foundry environment. For more information, see Requesting an Additional Virtual Host in Cloud Foundry Environment [page 46].</td>
<td>New</td>
<td>2020-08-11</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Proxy States</td>
<td>As an API Management administrator, you can set states for an API proxy while creating or updating the API proxy. For more information, see API Proxy States [page 393].</td>
<td>New</td>
<td>2020-08-11</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>OpenAPI Specification 3.0 in API Management</td>
<td>API Management now supports OpenAPI Specification (OAS) 3.0. For more information, see OpenAPI Specification 3.0 [page 82].</td>
<td>New</td>
<td>2020-08-11</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Capability</td>
<td>Environment</td>
<td>Title</td>
<td>Description</td>
<td>Type</td>
<td>Available as of</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Migration Assistance for API Management from Neo to Cloud Foundry Environment</td>
<td>You can now choose to migrate your API Management artifacts from an existing API Management subscription in the Neo environment to another API Management subscription in the public cloud infrastructures (hyperscalers) within the Cloud Foundry environment. For more information, see Migration Assistance for API Management from Neo to Cloud Foundry Environment [page 624].</td>
<td>New</td>
<td>202-08-06</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Access plan for API business hub enterprise</td>
<td>Take a look at the newly introduced API Access plan for the API business hub enterprise in the Cloud Foundry environment. Creating a service instance using this plan enables you to use APIs to interact with the API business hub enterprise. For more information, see API Access plan for API business hub enterprise [page 59].</td>
<td>New</td>
<td>202-07-23</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Access plan for API Portal</td>
<td>Take a look at the newly introduced API Access plan for the API Portal in the Cloud Foundry environment. Creating a service instance using this plan enables you to use APIs to interact with the API Portal. For more information, see API Access plan for API Portal [page 56].</td>
<td>New</td>
<td>202-07-23</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>On-premise Connectivity plan</td>
<td>Introduced On-Premise Connectivity plan in the Cloud Foundry environment. Creating a service instance using this plan helps you to obtain a service key to enable principal propagation. For more information, see On-Premise Connectivity Plan [page 54].</td>
<td>New</td>
<td>202-06-04</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Provider of type On Premise</td>
<td>Create an API Provider of type On Premise to connect to an on-premise system via Cloud Connector. For more information, see Create an API Provider [page 374]</td>
<td>New</td>
<td>202-06-04</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Management as Route Service plan</td>
<td>Introduced API Management as Route Service plan in the Cloud Foundry environment. Creating a service instance using this plan helps you in managing the Cloud Foundry applications. For more information, see Route Service plan [page 51].</td>
<td>New</td>
<td>202-06-04</td>
</tr>
</tbody>
</table>
## Technical Component

<table>
<thead>
<tr>
<th>Component</th>
<th>Capability</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Type</th>
<th>Available as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>API Provider of type Open Connectors</td>
<td>You can now create an API Provider of type Open Connectors to connect to third-party APIs. For more details, see Create an API Provider [page 374]</td>
<td>New</td>
<td>202-06-04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Creating Custom Role</td>
<td>You can now create a custom role for API products in API Management. Take a look at Creating a Custom Role [page 67] to know more. You can also assign permission to a product via UI. For more information, see Assign Permission to a Product via UI [page 487].</td>
<td>New</td>
<td>202-03-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Unsubscribing to the API portal and API business hub enterprise applications</td>
<td>If necessary, users can unsubscribe to the API portal and API business hub enterprise applications. For more details, see Cancelling API Management Service Subscription [page 72].</td>
<td>New</td>
<td>202-03-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Setting up API Portal and API business hub enterprise applications</td>
<td>You can now set up your API Portal and API business hub enterprise applications on Cloud Foundry environment. Take a look at the initial setup of API Portal application Setting Up API Portal Application [page 43]. For setting up the API business hub enterprise application, see Setting Up API business hub enterprise Application [page 49].</td>
<td>New</td>
<td>202-02-28</td>
</tr>
</tbody>
</table>

**Parent topic:** Archive - Release Notes for SAP API Management [page 16]

## Related Information

- 2021 Archives [page 17]
- 2019 Archives [page 29]
- 2018 Archives [page 33]
- 2017 Archives [page 35]
- 2016 Archives [page 37]
### 1.2.1.3 2019 Archives

<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Capability</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Type</th>
<th>Available as</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Automated setup of API Management service</td>
<td>We have now completely automated the provisioning process of setting up an API Management service using . For more information see .</td>
<td>New</td>
<td>2019-1 2-23</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>The Updated UI of API Portal and API business hub enterprise with enhanced visual experience.</td>
<td></td>
<td>New</td>
<td>2019-1 2-10</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Restoration of Application ID across Landscapes</td>
<td>As a user with AuthGroup.API.Admin role, you can now maintain the same application ID across API Management landscapes. For more information refer Restoring Application ID across SAP API Management Landscapes [page 475]</td>
<td>New</td>
<td>2019-1 0-24</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Custom Attributes</td>
<td>Introduced custom attributes at the application level. As an admin, you can create applications on behalf of other application developers, add custom attributes to applications. For more information refer</td>
<td>New</td>
<td>2019-1 0-04</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Microsoft Azure Services</td>
<td>Now, starter plan allows you to directly enable API Management on another public cloud infrastructure, Microsoft Azure Services. Check to see how to get started.</td>
<td>New</td>
<td>2019-0 9-30</td>
</tr>
</tbody>
</table>
### Technical Component

<table>
<thead>
<tr>
<th>Capability</th>
<th>Environment</th>
<th>Title</th>
<th>Description</th>
<th>Type</th>
<th>Available as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Manageme nt Integration Suite Neo Cloud Foundry Sunset of Classic API business hub enterprise</td>
<td></td>
<td></td>
<td>The Classic Design API business hub enterprise is no longer available. You can continue to use the New Design API business hub enterprise.</td>
<td>New</td>
<td>2019-09-05</td>
</tr>
<tr>
<td>API Management Integration Suite Neo Cloud Foundry</td>
<td></td>
<td></td>
<td>SAP API Management 1.0, client SDK is now available. You can download the Client SDK package by navigating to API Portal and choosing Client SDK. For more information, see</td>
<td>New</td>
<td>2019-09-05</td>
</tr>
<tr>
<td>API Management Integration Suite Neo Cloud Foundry</td>
<td></td>
<td></td>
<td>Introduced JSON Web Token policy. As an admin, you can now attach a JWT policy to an API Proxy. For more information refer to JSON Web Tokens [page 180]</td>
<td>New</td>
<td>2019-08-12</td>
</tr>
<tr>
<td>API Management Integration Suite Neo Cloud Foundry</td>
<td></td>
<td></td>
<td>Introduced custom attributes. As an admin, you can now add custom attributes to a product. For more information, see Custom Attributes [page 468].</td>
<td>New</td>
<td>2019-08-01</td>
</tr>
<tr>
<td>API Management Integration Suite Neo Cloud Foundry</td>
<td></td>
<td></td>
<td>Introduced navigation categories section in the API business hub enterprise. As an Content Admin, you can now configure categories. Configured categories are visible on the home page. For more information, see Manage Navigation Categories [page 510].</td>
<td>New</td>
<td>2019-07-23</td>
</tr>
<tr>
<td>API Management Integration Suite Neo Cloud Foundry</td>
<td></td>
<td></td>
<td>Introduced ‘content admin’ role for API business hub enterprise. As an API admin, you can now assign AuthGroup.Content.Admin role to users. Users assigned with this role can configure categories. For more information, see</td>
<td>New</td>
<td>2019-07-23</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Capability</td>
<td>Environment</td>
<td>Title</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Policy Attributes</td>
<td>Updates to XML to JSON policy and Message Logging policy. Added a new attribute <code>&lt;TreatAsArray&gt;</code> to XML to Json policy. <code>&lt;SSLInfo&gt;</code> element added to Message Logging policy. For more information, see XML to JSON [page 270] and Message Logging Policy [page 213].</td>
<td>Update</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Cloud Foundry</td>
<td>Advanced API Analytics</td>
<td>Advanced API Analytics brings to you the all new analytics dashboard, providing handy and powerful analytical reporting tools to track your API performance and usage. Fore more information, see .</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Advanced API Hub Enterprise User Roles</td>
<td>Introduced 'site admin' role for API business hub enterprise and updated existing API business hub enterprise roles. As an API admin, you can now assign AuthGroup.Site.Admin role to users. Users assigned with this role can configure updates and perform portal changes. For more information, see .</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Custom Attributes</td>
<td>API Management now supports custom attributes for applications and products. These attributes can be accessed during API call via verify api key, access token, and access entity policies. For more information, see Custom Attributes [page 468].</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Open Connector Policy</td>
<td>Enhancements to open connector policy. You can now add a service callout policy to an open connector type API. For more information, see Open Connectors [page 308]</td>
<td>Changed</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Managing API Business Hub Enterprise</td>
<td>Introduced updates section in the API business hub enterprise. As an API admin, you can now configure updates. Configured updates is visible to the application developers in the updates section. For more information, see Manage Updates [page 509].</td>
<td>Changed</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Key Value Map</td>
<td>Updates to KVM service. As an API admin, you can now create, update, and delete KVM at API Proxy level. For more information, see .</td>
<td>Changed</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>API Proxy</td>
<td>You can now create an API Proxy from an Open Connector type API Provider. For more information, see Create an API from the API Portal [page 393]</td>
<td>Changed</td>
</tr>
<tr>
<td>Technical Component</td>
<td>Capability</td>
<td>Environment</td>
<td>Title</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Open Connectors</td>
<td>Open Connectors type connection is now available in API Provider. Open Connectors provides pre-built connectors to simplify the connectivity and seamless integration with non-SAP cloud applications. Open Connectors enable you to connect to third-party APIs via harmonized RESTful APIs. For more information, see Create an API Provider [page 374]</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Key Value Map</td>
<td>You can now create, update, and delete key value maps on the API Portal. To do so, on the home screen select Configure Key Value Map. For more information, see Configure a Key Value Map [page 462].</td>
<td>Changed</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Synchronize API</td>
<td>This functionality updates the API with the latest version available in the back end. For more information, see Create an API form API Portal [page 393]</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>API business hub enterprise Next Generation design is now available.</td>
<td>API Management API business hub enterprise Next Generation design is now available.</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>My Workspace</td>
<td>Application developers can now navigate to My Workspace in the API business hub enterprise to create applications, subscribe to products, and view cost and analytics.</td>
<td>New</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Manage the API business hub enterprise</td>
<td>As an API admin, navigate to Manage to configure the API business hub enterprise and manage user access to the portal.</td>
<td>Changed</td>
</tr>
<tr>
<td>API Management</td>
<td>Integration Suite</td>
<td>Neo Cloud Foundry</td>
<td>Test Console</td>
<td>Test the APIs by navigating to Test Console in the API business hub enterprise.</td>
<td>Changed</td>
</tr>
</tbody>
</table>

Parent topic: Archive - Release Notes for SAP API Management [page 16]
## 1.2.1.4 2018 Archives

All the features released in 2018 are archived here.

<table>
<thead>
<tr>
<th>Function</th>
<th>Type of Change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API resource</td>
<td>Update</td>
<td>API resource documentation is now rendered in swagger 3.0.</td>
</tr>
<tr>
<td>Key Value Map</td>
<td>Update</td>
<td>You can now add a new entry to an existing key value map. For more information, see <a href="#">API Management on Cloud Foundry</a></td>
</tr>
<tr>
<td>API Management on Cloud Foundry</td>
<td>Update</td>
<td>Apart from <strong>preview</strong> and <strong>lite</strong>, you can now choose a new subscription plan <strong>starter</strong> for enabling API Management on Cloud Foundry. <strong>Starter</strong> plan allows you to directly enable API Management and is available on a public cloud infrastructure, Amazon Web Services.</td>
</tr>
<tr>
<td>Creating an API Proxy</td>
<td>Update</td>
<td>As an API admin, you can now use an existing API Proxy to create an API Proxy. For more information, see <a href="#">Creating an API Provider</a></td>
</tr>
<tr>
<td>Creating an API Provider</td>
<td>Update</td>
<td>As an API admin, you can now specify the sap-client in the API Provider. You can specify the sap-client, either by updating an existing API Provider or by creating an API Provider. For more information, see <a href="#">Key Value Map</a></td>
</tr>
<tr>
<td>Key Value Map</td>
<td>Update</td>
<td>Enhancements to KVM GET service. For more information, see <a href="#">Creating an API Proxy</a></td>
</tr>
<tr>
<td>Updating virtual host alias</td>
<td>Update</td>
<td>As an API admin, you can now edit an API Proxy to update the virtual host alias.</td>
</tr>
<tr>
<td>Service Callout policy</td>
<td>Update</td>
<td>You can now refer an existing API in Service Callout Policy. For more information, see <a href="#">Service Callout policy</a></td>
</tr>
<tr>
<td>Assigning permissions to a product</td>
<td>Update</td>
<td>As an API Admin, you can provide permission via service to user roles for discovering and subscribing to a product in the API business hub enterprise. For more information, see <a href="#">Assigning permissions to a product</a></td>
</tr>
<tr>
<td>Policy editing</td>
<td>Update</td>
<td>In the policy editor, you can now edit the policy name.</td>
</tr>
<tr>
<td>Function</td>
<td>Type of Change</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Copying an API Proxy</td>
<td>New</td>
<td>As an API Portal admin, you can now copy an existing API in the same subscription and deploy the copied API. For more information, see</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adding key value map</td>
<td>Update</td>
<td>You can now create secure key value maps to save encrypted values. This can be done by setting the ‘encrypted’ field as ‘true’, while creating a key value map. For more information, see</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete custom charts</td>
<td>Update</td>
<td>You can now delete the custom charts that you have created. To do so, navigate to Analyze Custom View and then select the chart that you want to delete.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edit custom charts</td>
<td>Update</td>
<td>You can now edit a custom chart that you have created. To do so, navigate to Analyze Custom View and then select the chart that you want to edit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSI info in Service Callout policy</td>
<td>Update</td>
<td>You can now enable SSL in service callout policy. For more information, see</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API resource documentation in the API</td>
<td>Update</td>
<td>API resources documentation is now available in Swagger rendered UI format in the API business hub enterprise.</td>
</tr>
<tr>
<td>business hub enterprise</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product publishing</td>
<td>Update</td>
<td>As an API Admin, you can now delete permission to user roles for discovering and subscribing to a product in the API business hub enterprise. Only users assigned with the required role can discover and subscribe to the product.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API resource documentation in the API</td>
<td>Update</td>
<td>API resources documentation is now available in Swagger rendered UI format in the API Portal. To view the documentation in the API Portal, navigate to Develop-&gt;API-&gt;Resource.</td>
</tr>
<tr>
<td>Portal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product publishing</td>
<td>Update</td>
<td>As an API Admin, you can provide permission to user roles for discovering and subscribing to a product in the API business hub enterprise. Only users assigned with the required role can discover and subscribe to the product. For more information, see</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APIs on Discovery service</td>
<td>New</td>
<td>Discovery service APIs are now available on SAP API Business Hub. For more information, see SAP API Business Hub.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enabling principal propagation for on</td>
<td>New</td>
<td>As an API Admin, you can now enable principal propagation to your on premise backend. For more information, see</td>
</tr>
<tr>
<td>premise backend.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Parent topic:** Archive - Release Notes for SAP API Management [page 16]

**Related Information**

2021 Archives [page 17]
## 1.2.1.5 2017 Archives

All the features released in 2017 are archived here.

<table>
<thead>
<tr>
<th>Function</th>
<th>Type of Change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy template</td>
<td>New</td>
<td>You can now create and apply policy templates to an API Proxy. For more information, see</td>
</tr>
<tr>
<td>Policy type</td>
<td>New</td>
<td>You can now add new policies to an API Proxy. For more information, see and</td>
</tr>
</tbody>
</table>
| Metering data             | New            | • You can now view the number of calls made for all APIs in a product. For more information, see  
                           | • You can now view the number of calls made for all APIs in an application. For more information, see   
<pre><code>                       | • You can now view the number of calls made by the application key for the current month. For more information, see   |
</code></pre>
<p>| Policy template           | New            | You can now import, export, and delete policy templates to an API Proxy. For more information see , and |
| Monetization              | New            | You can now create rate plans and attach rate plans to a product in the API Portal. View bill details in the API Portal and the API business hub enterprise. For more information see,   |
| Policy template           | New            | You can now update a policy template using API Portal. For more information, see |
| Key Value Map             | Update         | You can now update key value map entry. For more information, see             |
| Refresh                   | New            | A refresh button is introduced from this release. The refresh button is available in the Develop screen of API Portal and in the Consume screen of API business hub enterprise. |
| Selecting host alias      | New            | While creating an API, you can now select a virtual host alias from the list of available host aliases. For more information, see   |
| Creating multiple virtual host | New           | You can now create a new virtual host or update an alias for an existing virtual host. For more information, see   |
| Billing                   | Update         | You can now download bill details from API Portal and API business hub enterprise. For more information, see and |
| Updating and Deleting rate plans | New       | You can now update or delete a rate plan from API Portal For more information, see and |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Type of Change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selecting Virtual Host</td>
<td>Update</td>
<td>You can now select a virtual host alias while importing an API. For more information, see</td>
</tr>
<tr>
<td>Multiple cloud connectors connecting to one subscriber account</td>
<td>New</td>
<td>For multiple cloud connector configurations, you can now choose a particular cloud connector by providing the location ID during creation of an API Provider. For more information, see</td>
</tr>
<tr>
<td>Discover API packages in API Portal</td>
<td>New</td>
<td>You can now view SAP API Management platform supported API packages available in SAP Content Hub on the API Portal. For more information, see</td>
</tr>
<tr>
<td>Selecting host alias</td>
<td>New</td>
<td>While creating an API, you can now select a virtual host alias from the list of available host aliases. For more information, see</td>
</tr>
<tr>
<td>Copying policy template</td>
<td>New</td>
<td>Using the copy feature available in Discovery page, you can now copy policy templates available in SAP API Business Hub on to API Portal. For more information, see</td>
</tr>
<tr>
<td>Tier based rate plan</td>
<td>New</td>
<td>You can now create tier based rate plans. Tier based rate plan allows you to create tiers or API call ranges and rate charged per API call varies from one tier to another tier. For more information, see</td>
</tr>
<tr>
<td>Copying APIs</td>
<td>New</td>
<td>Using the copy feature available in Discovery page, you can now copy APIs available in SAP API Business Hub on to API Portal. For more information, see</td>
</tr>
<tr>
<td>Copying policy template</td>
<td>Update</td>
<td>Using the copy feature available in Discovery page, you can now choose a policy template artifact and from the policy template details page copy the policy template to your API Portal. For more information, see</td>
</tr>
<tr>
<td>Copying APIs</td>
<td>Update</td>
<td>Using the copy feature available in Discovery page, you can now choose an API artifact and from the API details page copy the API to your API Portal. For more information, see</td>
</tr>
<tr>
<td>View in API Business Hub</td>
<td>New</td>
<td>At present, at the package level you can view and copy APIs to API Portal. But, if you want to perform some actions like trying out the API or generating the code, then you need to navigate to SAP API Business Hub. You can do it by selecting View in API Business Hub. On selecting the link, you are navigated to the same package available in SAP API Business Hub. For more information, see</td>
</tr>
<tr>
<td>API Management as a service on Cloud Foundry</td>
<td>New</td>
<td>SAP API Management service is now discoverable from the SAP Cloud Foundry marketplace. Now you can</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• create an API Management service instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• delete an API Management service instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• bind a Cloud Foundry application to an API Management service instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• unbind a Cloud Foundry application from an API Management service instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see</td>
</tr>
</tbody>
</table>
### Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Type of Change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updating an API Management service instance on Cloud Foundry</td>
<td>New</td>
<td>You can now update an API Management service instance on Cloud Foundry. For more information, see the <strong>Updating an API Management Service Instance</strong> section.</td>
</tr>
<tr>
<td>Service to view Application Developer Details on API business hub enterprise</td>
<td>New</td>
<td>As an application developer, you can now view your personal data stored in the API business hub enterprise. For more information, see</td>
</tr>
</tbody>
</table>

**Parent topic:** Archive - Release Notes for SAP API Management [page 16]

### Related Information

- 2021 Archives [page 17]
- 2020 Archives [page 24]
- 2019 Archives [page 29]
- 2018 Archives [page 33]
- 2016 Archives [page 37]

### 1.2.1.6 2016 Archives

All the features released in 2016 are archived here.

#### API Management Design time

<table>
<thead>
<tr>
<th>Function</th>
<th>Type of Change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics Collector Policy</td>
<td>New</td>
<td>With this new policy, you can collect statistics for data in a message, such as product ID, price, REST action, client and target URL, and message length. For more information, see</td>
</tr>
<tr>
<td>Import of Swagger json file type</td>
<td>Enhanced</td>
<td>You can now import API content in swagger json file type. For more information, see</td>
</tr>
<tr>
<td>Update API Proxy</td>
<td>Enhanced</td>
<td>You can now update the API proxy at the time of importing an existing API proxy content. For more information, see</td>
</tr>
<tr>
<td>Implement Route rule on API Proxy</td>
<td>New</td>
<td>You can now implement route rule on API proxy to determine the path of a request from the ProxyEndpoint to the TargetEndpoint. For more information, see</td>
</tr>
<tr>
<td>Function</td>
<td>Type of Change</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Custom date range</td>
<td>Enhanced</td>
<td>You can now filter analytics data in custom date range. For more information, see</td>
</tr>
<tr>
<td>Set scope while defining API products</td>
<td>New</td>
<td>You can now set the scope at product level for restricting the access of authorization token on each application. For more information, see</td>
</tr>
<tr>
<td>Import of proxies from on-premise to on-demand</td>
<td>New</td>
<td>You can now import APIs in .zip file from API runtime format to API Management. For more information, see</td>
</tr>
<tr>
<td>Custom Charts</td>
<td>New</td>
<td>You can now create custom charts based on your preferred parameters. For more information, see</td>
</tr>
<tr>
<td>Detailed usage statistics: Minute-based usage metrics for the past hour</td>
<td>New</td>
<td>You can now filter charts with minute-based usage metrics for the past hour.</td>
</tr>
<tr>
<td>Configure cache resource</td>
<td>New</td>
<td>You can now configure cache resource at environment level and use them during policy definition. For more information, see</td>
</tr>
<tr>
<td>Publish products associating with API resources</td>
<td>Enhanced</td>
<td>You can now associate API resources with the published products. For more information, see</td>
</tr>
<tr>
<td>Add key value pairs at the environment level</td>
<td>New</td>
<td>You can now add key value pairs at environment level to manage API proxies scopes for longer-term data persistence. For more information, see</td>
</tr>
<tr>
<td>Add filters to custom charts</td>
<td>New</td>
<td>You can now add custom charts in Analytics dashboard. For more information, see</td>
</tr>
<tr>
<td>View applications from API portal</td>
<td>New</td>
<td>You can now view the applications in API Portal. For more information, see</td>
</tr>
<tr>
<td>Navigate to chart details from dashboard</td>
<td>Enhanced</td>
<td>You can now navigate to chart details from dashboard. For more information, see</td>
</tr>
<tr>
<td>API Portal home page</td>
<td>New</td>
<td>Once you log on to the API Portal you can now view a new home page. It shows tiles on various parameters at a high level of the current API Management account. Tiles include API Calls, API Errors, Total APIs, Applications, API Provider errors, consumption pattern of APIs and API provider, Products, and Developers.</td>
</tr>
<tr>
<td>View API Key and Secret</td>
<td>New</td>
<td>If you have logged on to the API Portal with the role APIPortal.Administrator, you can view the API key and secret of the applications.</td>
</tr>
<tr>
<td>UI improvements</td>
<td>Enhanced</td>
<td>The look and feel of the API Portal has been enhanced to provide better usability.</td>
</tr>
<tr>
<td>Save and publish products</td>
<td>New</td>
<td>You can now explicitly save and publish products. For more information, see</td>
</tr>
<tr>
<td>Function</td>
<td>Type of Change</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Configure custom charts based on multiple measures and dimensions</td>
<td>New</td>
<td>You can now configure charts based on multiple measures and dimensions, by setting the filters at the time of configuration. For more information, see</td>
</tr>
<tr>
<td>Regenerate application Key</td>
<td>Enhanced</td>
<td>You can now regenerate the application key and use the same in all your new requests. For more information, see</td>
</tr>
<tr>
<td>Manage API Provider certificates</td>
<td>New</td>
<td>You can now upload new API Provider certificates and associate them to the API providers at the time of registration. For more information, see</td>
</tr>
<tr>
<td>UI enhancements</td>
<td>New</td>
<td>UI has been enhanced to adapt Fiori based framework. All the navigation bars on the left-hand side of the application has been replaced with icons for better look and feel.</td>
</tr>
<tr>
<td>Define drill-down options in custom charts</td>
<td>Enhanced</td>
<td>You can now drill down in custom charts based on default or custom dimensions, so that you can analyze a specific dimension as required. For more information, see</td>
</tr>
<tr>
<td>Simplified Onboarding Process</td>
<td>New</td>
<td>You can now perform self-service based on-boarding of API Portal and Dev Portal from the HCP service cockpit. For more information, see</td>
</tr>
<tr>
<td>UI enhancements</td>
<td>New</td>
<td>UI has been enhanced to adapt Fiori based framework. Tabular view has been enabled with row wise actions in the following screens: API, API Provider, Certificate, Product, and Application.</td>
</tr>
<tr>
<td>Support for four new static dimensions</td>
<td>Enhanced</td>
<td>You can now filter analytics data based on the following four new static dimensions: Request URI, API Provider Host, Proxy Base Path and HTTP status code.</td>
</tr>
<tr>
<td>Breadcrumb option added in Analytics charts</td>
<td>New</td>
<td>When you apply filter on a chart to drill down in to details, you can navigate back to any previous parameter using the breadcrumb option. For more information, see</td>
</tr>
<tr>
<td>Create API from API designer</td>
<td></td>
<td>You can create APIs in open API format using the API designer that is made available on API Portal. For more information, see</td>
</tr>
<tr>
<td>SOAP Message Validation Policy</td>
<td>New</td>
<td>A new policy has been introduced to validate SOAP messages and reject the messages if it does not conform to the specified requirements. For more information, see</td>
</tr>
<tr>
<td>On Premise Connectivity</td>
<td>New</td>
<td>You can now connect to the on-premise system through cloud connector and create API provider of on-premise type. For more information, see</td>
</tr>
</tbody>
</table>
### Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Type of Change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post Client flow</td>
<td>New</td>
<td>You can now add a PostClientFlow to the ProxyEndpoint, which executes after the response is returned to the requesting client application. For more information, see</td>
</tr>
</tbody>
</table>

**Parent topic:** Archive - Release Notes for SAP API Management [page 16]

### Related Information

- 2021 Archives [page 17]
- 2020 Archives [page 24]
- 2019 Archives [page 29]
- 2018 Archives [page 33]
- 2017 Archives [page 35]

### 1.3 Patch Releases for API Management

This topic provides information on patch releases for API Management that are provided for bug fixes.

#### October 2021

**Software Version:** 1.135.*

<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Software Version</th>
<th>Description</th>
<th>Available as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>1.135.3</td>
<td>Error messages were popping up on the API Portal <strong>Onboarding settings</strong> page, and the page was not accessible. Backend configurations were applied to resolve this issue.</td>
<td>2021-10-2</td>
</tr>
</tbody>
</table>

#### September 2021

**Software Version:** 1.134.*
<table>
<thead>
<tr>
<th>Technical Component</th>
<th>Software Version</th>
<th>Description</th>
<th>Available as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>1.134</td>
<td>The following issues have been fixed with this patch:</td>
<td>2021-08-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• While importing a certificate, the common name of the root certificate was displayed in the portal instead of the common name of the certificate. The data inconsistency encountered by the users while importing certificates has been resolved.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When multiple IDPs were configured, users couldn’t approve the new developer registration requests in their Production environment, if the user ID of the developers weren’t their email IDs. With this patch, such restrictions on the User ID have been removed, and the users can approve the new developer registration requests in their Production environment.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.133.7</td>
<td>The deployment of an API Proxy would get timed out and fail when large number of Products were associated with it. This patch ensures that in such exceptional cases the deployment of the API Proxy wouldn’t fail due to timeout.</td>
<td>2021-08-1</td>
</tr>
<tr>
<td></td>
<td>1.133.6</td>
<td>The following issues have been fixed with this patch:</td>
<td>2021-08-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Developer registrations in API Business Hub Enterprise would fail when multiple IDPs were associated with the API Management subaccount, and the user ID of the user being onboarded was alphanumeric and not an email ID.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The deployment of API Proxies would fail, if the API Proxies in the published Products had access control permissions defined.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• API Proxy POST/PUT requests with gzip compressed content would fail.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Import of API Proxy would fail if it’s chained with another API Proxy. To resolve this issue, schema validation for the API Proxy chaining import flow has been enabled.</td>
<td></td>
</tr>
</tbody>
</table>
1.4 Configure

Before SAP API Management can be used, the tenant (customer) administrator provisions the API Management applications for the users.

This section includes links to concepts and activities required to set up and start using API Management as a service on Cloud Foundry.

1.4.1 Initial Setup

Includes links to concepts and activities required to set up and start using API Management as a service on Cloud Foundry.

SAP provisions the API Management applications for customers. The two applications that are provisioned are the API portal and API business hub enterprise.

- **Note**
  - All activities in this section are performed by the tenant (customer) administrator.
  - To subscribe to multitenant application from the Subscriptions page in the SAP BTP cockpit, see [Subscribe to Multitenant Applications in the Cloud Foundry Environment Using the Cockpit](#).
  - During the onboarding process, the tenant (customer) should expect API calls from the SAP API Management operations team. These calls may be manual or automated and are made to ensure completeness of the onboarding process.

This section covers the following activities:

- Setting Up API Portal Application [page 43]
- Setting Up API business hub enterprise Application [page 49]
- Account Members [page 63]
- Assigning User Roles [page 64]

- **Remember**
  The Cloud Foundry setup is different from the NEO setup. The tutorials apart from the setup remains same.

**Related Links:**

- Availability of SAP BTP Regions and Services
1.4.1.1 Setting Up API Portal Application

To create APIs, products, import policy templates, and view applications, set up the API portal application.

Context

Depending upon the license you hold, you can set up the API Management, API Portal capability from the Integration Suite launchpad, or you can use the standalone tile for API Management, API Portal to subscribe to the API Portal application.

Note

- You should either have Integration Suite subscription or API Management, API Portal tile visibility to set up the API Portal application.
- If you’re subscribing to the API Management, API portal in the Cloud Foundry subaccount, either by using the Integration Suite launchpad or the API Management, API portal standalone tile, ensure that you don’t have an instance of starter plan created in the same subaccount.
- API Management capabilities from Integration Suite and API Management subscriptions can’t coexist in a subaccount.

Procedure

1. In your web browser, log on to SAP BTP Cockpit and navigate to your subaccount.
2. You can set up the API Management, API Portal using one of the following methods:
   - **Method 1**: Activate the API Management, API Portal capability using the Integration Suite launchpad.
     1. In the navigation area of the subaccount, choose Services ➤ Service Market Place ➤.
     2. Search for Integration Suite and choose Create in the overview page. See Subscribing to Integration Suite for details.
     3. In the Integration Suite launchpad, under Capabilities section, choose Manage Capabilities. See Working with Integration Suite Launchpad, for more information.
     4. For API Portal subscription, choose Add Capabilities and select Design, Develop and Manage APIs checkbox.
     5. Choose Next.
     6. Select the Enable API Business Hub Enterprise checkbox if you want to activate API Business Hub Enterprise.
     7. Choose Activate.
   - **Method 2**: Subscribe to the API portal using the API Management, API portal standalone tile.
     2. Select the Standard plan.
     3. Choose Create under Actions.
After subscribing to the API Management, API portal application, don’t immediately choose Go to Application. To access the application, user must have the APIManagement.Selfservice.Administrator role. The steps for assigning the role are explained further.

3. Navigate to your subaccount, create a new role collection, add APIManagement.Selfservice.Administrator role to a role collection, and then assign the role to yourself. See Working with Role Collections for detailed instructions.

During the API portal setup, it’s recommended that you assign only the APIManagement.Selfservice.Administrator role.

4. Choose Go to Application from the API portal Subscriptions page or on the API Management, API portal tile.

If you have chosen Go to Application without assigning the APIManagement.Selfservice.Administrator role, then you receive an application authentication error. In such cases, assign the APIManagement.Selfservice.Administrator role to the user. If the error persists after assigning the role, clear your web browser cache, and log out of the application and log on again.

5. A Configure the API Management Service screen appears. Perform the following steps:
   a. In the Account section, select the Account Type.
      - Select Non-Production account type for, nonbusiness critical activities, integrating test systems, testing new scenarios, performance testing, and sandbox activities.
      - Select Production account type for business critical usage, integrating production systems, and productive APIs.
   b. Provide an e-mail ID in the Notification Contact field to receive updates. You can also add multiple e-mail IDs. To add more than one e-mail ID, press the Tab button on the keyboard.
   c. In the Virtual Host section, enter the Host Alias.
      Virtual hosts are the base URLs of an API Proxy that identifies your organization. Once you enter the host alias, your API Proxy would appear in the default API Management domain as shown here:
   d. Select the Make <User ID> API Portal administrator checkbox to assign the APIPortal.Administrator role to the mentioned user. Assigning administrator role helps the user to access API portal user interface and services.


7. In the Set-up Confirmation window, review the provided details and choose Confirm to start the onboarding process.

You’re now navigated to a progress window, which states API Management Service Setup In Progress. The Configuration process is triggered, where the necessary resources are provisioned for you, followed by Testing the Setup process, where a simple API Proxy is deployed and invoked to check that everything is set up properly. As and when the processes complete the indicators turn green to indicate that the processes
are successful. Then, a Release Notification mail is sent out to the e-mail ID/s provided in the Configure the API Management Service screen. This e-mail contains details of the newly set up API Management service on your account.

Results

The API Portal is now configured. Log in to the API portal again and you can now create APIs, build API proxies as a service provider, or use APIs and other convenient services.

Next Steps

To start publishing the API Portal content, you must enable the API Business Hub Enterprise. Navigate to Onboarding Settings and choose Connection.

i Note

To publish the API portal content on the API Business Hub Enterprise located in the same subaccount, see Setting Up API business hub enterprise Application [page 49].

To publish the API portal content on the centralized API Business Hub Enterprise, follow the on-screen instructions and see Centralized API business hub enterprise [page 73].
1.4.1.1.1 Requesting an Additional Virtual Host in Cloud Foundry Environment

Create a new virtual host or update an alias for an existing virtual host in the Cloud Foundry environment.

Prerequisites

- You must have a service key for the `APIManagement.Selfservice.Administrator` role. To know more about creating a service key for accessing APIs in the API portal, see the Creating a Service Key section in API Access plan for API Portal [page 56].
- You have fetched a valid bearer token. To know more about obtaining a bearer token, see the Obtaining a Bearer Token section in API Access plan for API Portal [page 56].

Context

Virtual hosts are the base URLs of an API Proxy. They include the protocol (http or https), along with the hostname of the API proxy, which maps to a router address and port through DNS. You can configure multiple virtual hosts for a given subscription using the following steps:

- You can create a new virtual host.
- You can update an alias of an existing virtual host.

Procedure

1. Run the services using a standard REST console.
2. Service to create a new virtual host:

   ○ Service URL: https://<url-from-service-key>/apiportal/operations/1.0/Configuration.svc/VirtualHostRequests
○ Method: POST
○ Request Header: bearer token: fetch
○ Content Type: application/json
○ Request Body:

```
Sample Code

{
  "accountId": "subdomain of your subaccount",
  "virtualHostUrl": "testvh12",
  "isDefaultVirtualHostRequest": false,
  "operation": "CREATE"
}

<!-- accountId - your subdomain account ID
Note: Ensure that the sub-account ID is entered in lowercase.
virtualHostUrl - virtual host alias e.g prod-apis , testapi
isDefaultVirtualHostRequest - Value is true, If you want the new virtual host to be the default virtual host, else value is false -->
```

○ Response: 201

```
Sample Code

{
  "d": {
    "__metadata": {
      "id": 
      "url": 
      "type": "apimgmtconfiguration.VirtualHostRequest"
    },
    "accountId": 
    "allocatedPort": 
    "allocationStatus": "COMPLETE",
    "certStore": null, 
    "clusterName": 
    "id": 
    "isDefaultVirtualHostRequest": false, 
    "isForCustomDomain": false, 
    "isForNonSni": false, 
    "keyStoreAlias": null, 
    "keyStoreName": null, 
    "life_cycle": {
      "__metadata": {
        "type": "apimgmtconfiguration.History"
      },
      "changed_at": "",
      "changed_by": "",
      "created_at": "",
      "created_by": ""
    },
    "operation": "CREATE",
    "virtualHostId": 
    "virtualHostUrl": ""
  }
}
```

○ Create an incident on the component OPU-API-OD-OPS by navigating to https://support.sap.com to complete the configuration. Provide the VirtualHostRequest ID details in the ticket. You can retrieve the VirtualHostRequest ID from the location header.
3. Service to update an alias of an existing virtual host:
   ○ Service URL: https://<url-from-service-key>/apiportal/operations/1.0/Configuration.svc/VirtualHostRequests
   ○ Method: POST
   ○ Request Header: Bearer Token
   ○ Content Type: application/json
   ○ Request Body:

   ```json
   {
     "accountId": "<subdomain of your subaccount>",
     "virtualHostUrl": "<virtual host alias>",
     "isDefaultVirtualHostRequest": false,
     "operation": "UPDATE",
     "virtualHostId": "<id-of-an-existing-Virtual-Host-in-api-portal>"
   }
   ```

   Note: Ensure that the sub-account ID is entered in lowercase.

   virtualHostUrl - virtual host alias e.g prod-apis , testapi

   isDefaultVirtualHostRequest - Value is true, If you want the new virtual host to be the default virtual host, else false

   virtualHostId : Access the API from your browser: https://<url-from-service-key>/apiportal/operations/1.0/Configuration.svc/VirtualHostRequests

   Alternatively, invoke this API using the api portal api acess with an admin service key.

   To know more about creating a service key for accessing APIs in the API portal, see the Creating a Service Key section in .

   Example -->
   ```
   GET https://apiprotal-url/apiportal/api/1.0/Management.svc/VirtualHosts/
   ```

   ```json
   {
     "d": {
       "@odata.context": "https://apiportal-url/apiportal/api/1.0/Management.svc/$metadata",
       "results": [
         {
           "__metadata": {
             "id": "https://apiportal-url/apiportal/api/1.0/Management.svc/VirtualHosts('689333a2-2f6b-4029-8734-2b36a687e559')",
             "url": "https://apiportal-url/apiportal/api/1.0/Management.svc/VirtualHosts('689333a2-2f6b-4029-8734-2b36a687e559')",
             "type": "apiportal.VirtualHost"
           },
           "id": "689333a2-2f6b-4029-8734-2b36a687e559", <!-- required virtualHostId -->
           "isDefault": false,
           "isSSL": true,
           "life_cycle": {
             "__metadata": {
               "type": "apiportal.History"
             },
             "changed_at": "/Date(1487244078853)/",
             "changed_by": "X",
             "created_at": "/Date(1487244078853)/",
             "created_by": "X"
           },
           "name": "689333a2-2f6b-4029-8734-2b36a687e559",
           "projectPath": null,
           "virtual_host": "testvh12.dmzmo.sap.corp",
           "virtual_port": 443
         }
       ]
     }
   }
   ```
Response: 201
Create an incident on the component OPU-API-OD-OPS by navigating to https://support.sap.com to complete the configuration. Provide the VirtualHostRequest ID details in the ticket. You can retrieve the VirtualHostRequest ID from the location header.

**i Note**
After the virtual host is updated, APIs associated to a product using the updated virtual host must be redeployed and republished.

### 1.4.1.2 Setting Up API business hub enterprise Application

To discover, consume and monitor API from a centralized API catalog, set up the API business hub enterprise application.

**Context**

Depending upon the license you hold, you can set up the *API Business Hub Enterprise* capability from the *Integration Suite* launchpad, or you can use the standalone tile for *API Business Hub Enterprise* to subscribe to the API Business Hub Enterprise application.

**i Note**

- You should either have *Integration Suite* subscription or *API Business Hub Enterprise* tile visibility to set up API Business Hub Enterprise.
- If you’re subscribing to *API Business Hub Enterprise* in the Cloud Foundry subaccount, either by the using the *Integration Suite* launchpad or the *API Business Hub Enterprise* standalone tile, ensure that you don’t have an instance of starter plan created in the same subaccount.
- API Management capabilities from Integration Suite and API Management subscriptions can’t coexist in a subaccount.

**Procedure**

1. In your web browser, log on to SAP BTP Cockpit and navigate to your subaccount.
2. You can set up the *API Business Hub Enterprise* using one of the following methods:
   - **Method 1**: Activate the *API Business Hub Enterprise* capability using the *Integration Suite* launchpad.
     1. In the navigation area of the subaccount, choose [Services >> Service Market Place](#).
2. Search for Integration Suite and choose Create in the overview page. See Subscribing to Integration Suite for details.

3. In the Integration Suite launchpad, under Capabilities section, choose Manage Capabilities. See Working with Integration Suite Launchpad, for more information.

4. For API Portal subscription, choose Add Capabilities and select Design, Develop and Manage APIs checkbox.

5. Choose Next.


7. Choose Activate.

**Method 2:** Subscribe to the API Business Hub Enterprise using the API Business Hub Enterprise standalone tile.


2. Select the Standard plan.

3. Choose Create under Actions.

---

### Note

After subscribing to the API Business Hub Enterprise application, don’t immediately choose Go to Application. To access the application, user must have the APIManagement.Selfservice.Administrator role. The steps for assigning the role are explained further.

---

3. Navigate to your subaccount, create a new role collection, add APIManagement.Selfservice.Administrator role to a role collection, and then assign the role to yourself. See Working with Role Collections for detailed instructions.

---

### Note

Use APIManagement.Selfservice.Administrator role during the onboarding of API Business Hub Enterprise and to get access to it.

---

4. Choose Go to Application from the API business hub enterprise Subscriptions page or on the API Management, API business hub enterprise tile.

---

### Note

If you have chosen Go to Application without assigning the Authgroup.SelfService.Admin role, then you receive an application authentication error. In such cases, assign the Authgroup.SelfService.Admin role to the user. If the error persists after assigning the role, clear your web browser cache, and log out of the application and log on again.

---

### Note

An email notification is sent to the logged in user regarding the setup.

---

### Related Information

Consume APIs [page 503]
1.4.1.3 API Management, API portal as a Service

The API Management, API portal as a service on Cloud Foundry provides different capabilities through Route Service plan, On-Premise Connectivity plan, and API Access plan.

The API Management, API portal supports the following plans:

- **apim-as-route-service**: The API Management as Route Service plan helps you in managing Cloud Foundry applications by including policies such as rate limit, quota. The service instance you create through this plan allows you to bind to the route service and creates an API Proxy. This API Proxy serves in establishing a secure connection with your Cloud Foundry application and all the calls made to the Cloud Foundry application are routed via API Management, API portal. For more details, see Route Service plan [page 51].

- **on-premise-connectivity**: The On-Premise Connectivity plan helps in achieving principal propagation while connecting to an on-premise backend system. To accomplish principal propagation you require a service key and this plan allows you to obtain a service key by creating a service instance. For more details on On-premise connectivity plan, see On-Premise Connectivity Plan [page 54].

- **apiportal-apiaccess**: The API Access plan allows you to generate a service key by creating a service instance. The service key, consisting of url (application url), clientId, clientSecret, and tokenUrl is used to generate a Bearer Token with the help of a REST Console. This Bearer Token, along with the application url and API endpoint are used to trigger the API. Therefore, Bearer Token acts like a key to access the APIs. For more details, see API Access plan for API Portal [page 56].

Related Information

Create an API Provider [page 374]

1.4.1.3.1 Route Service plan

The API Management as route service plan helps you in managing Cloud Foundry applications by including policies like rate limit, quota, and so on.

About the Plan

The service instance you create through this plan allows you to bind to the route service and creates an API Proxy. This API Proxy serves in establishing a secure connection with your Cloud Foundry application and all the calls made to the Cloud Foundry application are routed via API Management, API portal.
Prerequisites

You have subscribed to the API Management, API portal tile in the Cloud Foundry environment. For more information, see Setting Up API Portal Application [page 43].

You have the space developer role assigned to you.

Creating an API Management, API portal Service Instance

Create a service instance in API Management, API portal to start managing your Cloud Foundry applications.

Follow the below procedure to create a service instance on Cloud Foundry:

1. In your web browser, open the SAP BTP Cockpit - https://account.hana.ondemand.com/cockpit.
2. From your Subaccount, navigate to Spaces in your Cloud Foundry environment and choose Services Service Marketplace.
3. Choose API Management, API portal Instances New Instance.
4. In the Create Instance dialog, choose apim-as-route-service plan.
5. Choose Next until you reach the Confirm section.
6. In the section Confirm, enter a unique Instance Name and choose Finish.

*Note*

The apim-as-route-service plan allows you to create multiple service instances and connect to many Cloud Foundry applications using the same Subaccount.

Deleting an API Management, API portal Service Instance

Delete an API Management, API Portal service instance.

Prerequisites

You have the space developer role assigned to you.

You have created a service instance under API Management, API portal.

Use the following procedure to delete an API Management service instance on Cloud Foundry.

Procedure

1. In your Web browser, open the SAP BTP Cockpit.
2. In the provider account, choose Services Service Marketplace Instances.
3. Select the API Management, API portal tile.
4. Choose Instances from the left-hand pane.
5. From the list of instances visible, select the instance that you want to delete and choose  
6. Choose OK.
Binding a Cloud Foundry Application to an API Management, API portal Service Instance

Create a service instance and bind the Cloud Foundry application to API management, API portal service. When you bind an application, an API proxy is created and a new route is added to the application. The route initially redirects all calls to the proxy URL and then to the application.

Prerequisites

- You have the space developer role assigned to you.
- You have created a service instance under API Management, API portal.

Open the command-line interface for Cloud Foundry and enter the following command:

```
cf bind-route-service sap-cf-domain.com apim-service-instance-name --hostname my-app -c '{"api_name" : "custom_api_proxy_name"}'
```

--- Example

//Without parameters
cf bind-route-service cfapps.sap.hana.ondemand.com apim-prod-instance --hostname taxapp

//Cloud foundry URL for the above example is https://taxapp.cfapps.sap.hana.ondemand.com

//With parameters for Linux/MAC system
cf bind-route-service sap-cf-domain.com apim-service-instance-name --hostname my-app -c '{"api_name" : "test_api"}"

//With parameters for Windows system
cf bind-route-service sap-cf-domain.com apim-service-instance-name --hostname my-app -c "{"api_name" : "test_api"}"

---

**Note**

API Management, API portal supports only English alpha numerics, hyphens (-) and underscores (_) characters for "api_name".

You can bind an application to a service only from the command-line interface and not from SAP BTP Cockpit.

Providing a value for the parameter during binding is optional. If you provide a value for api_name, then the API proxy created in API Management, API portal for current binding gets the given name. Also, if an API with the same name exists in the API portal, then the same API proxy is used for the binding. That is, the API proxy end point is registered as the route service URL for the current binding.

Unbinding a Cloud Foundry Application from an API Management, API portal Service Instance

When you unbind a Cloud Foundry application from an API Management, API portal service instance, an API proxy is undeployed and the connection between the route service and the Cloud Foundry application is removed.
Prerequisites

- You have logged on as a space developer.
- You have bound an application to service instance under API Management, API portal.

In order to unbind, open the command prompt and enter the following command:

<table>
<thead>
<tr>
<th>Sample Code</th>
</tr>
</thead>
</table>
| cf unbind-route-service sap-cf-domain.com apim-service-instance-name --hostname my-app  
<-- Example  
 cf unbind-route-service cfapps.sap.hana.ondemand.com apim-prod-instance --hostname taxapp  
-->

<table>
<thead>
<tr>
<th>i Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can unbind an application from a service only from the command-line interface and not from SAP BTP Cockpit.</td>
</tr>
</tbody>
</table>

1.4.1.3.2 On-Premise Connectivity Plan

The On-premise connectivity plan helps in achieving principal propagation while connecting to an on-premise backend system.

About the Plan

To accomplish principal propagation, you require a service key. This plan allows you to obtain the token by creating a service instance and generating a service key.

This topic explains how to obtain a service key in order to enable principal propagation using API Management, API portal in the Cloud Foundry Environment.

Prerequisites

- You have subscribed to the API Management, API portal tile in the Cloud Foundry environment. For more information, see Setting Up API Portal Application [page 43].
- You have created an API Provider of type On Premise and chosen Principal Propagation as a mode of authentication to connect to an on-premise system. For more information, see Create an API Provider [page 374].
- You have the space developer role assigned to you.
Creating a Service Instance on API Management, API Portal

Create a service instance to generate a service key that is used to enable the principal propagation.

1. In your web browser, open the SAP BTP Cockpit - https://account.hana.ondemand.com/cockpit.
2. From your Subaccount, navigate to Spaces in your Cloud Foundry environment and choose Services > Service Marketplace.
3. Choose API Management, API portal > Instances > New Instance.
4. In the Create Instance dialog, choose on-premise-connectivity plan.
5. Click Next until you reach the Confirm section.
6. In the section Confirm, enter a unique Instance Name, and choose Finish.

After you have created a service instance, proceed with:

**Service Key Generation:**

1. Choose the created service instance link from the visible list.
2. In the left-hand pane, navigate to Service Keys > Create Service Key.
3. In the Create Service dialog, provide a Name and Description (optional).
4. Click Save. The client credentials like url, clientId, clientSecret, and tokenUrl details appear for the given instance name.

   Copy the client credentials in a notepad. You can now navigate back to Principal Propagation from Neo to the Cloud Foundry Environment or Principal Propagation from the Same Cloud Foundry Subaccount as you have the required client credentials.

You can use the credentials to establish:

- **Principal Propagation from the Neo to the Cloud Foundry Environment** [page 383]: Enable an application in your subaccount in the Neo environment to access an API Management proxy created on a Cloud Foundry based API Management, API portal without a user login. For this scenario to work, the Neo subaccount needs to be trusted by the Cloud Foundry subaccount where API Management, API portal is enabled. Now, the application on Neo can call API Management proxy using OAuth2SAMLBearer destination.

- **Principal Propagation from the Same Cloud Foundry Subaccount** [page 386]: Enable an application in your subaccount in the Cloud Foundry environment to access an API Management proxy created on the same Cloud Foundry based API Management, API portal without a user login. The JWT user token in your application can be exchanged with the API Management token using the service key credentials created for API Management, API portal.
1.4.1.3.3 API Access plan for API Portal

The API Access plan allows you to generate a service key by creating a service instance.

About the Plan

The service key, consisting of url (application url), clientId, clientSecret, and tokenUrl is used to generate a Bearer Token with the help of a REST Console. This Bearer Token, along with the application url and API endpoint are used to trigger the API. Therefore, Bearer Token acts like a key to access the APIs.

This topic explains how to enable API access for API Management, API portal.

Prerequisites

- You have subscribed to the API Management, API portal tile in the Cloud Foundry environment. For more information, see Setting Up API Portal Application [page 43].
- You have the space developer role assigned to you.

Creating a Service Instance in the API Management, API portal

Create a service instance using API Access plan to generate a service key.

1. In your web browser, open the SAP BTP Cockpit - https://account.hana.ondemand.com/cockpit.
2. From your Subaccount, navigate to Spaces in your Cloud Foundry environment and choose Services Service Marketplace.
3. Choose API Management, API portal Instances New Instance.
4. In the Create Instance dialog that opens, choose the apiportal-apiaccess plan.
5. In the section Specify parameters, paste one of the following JSON codes, to assign a specific role.

   The following roles are supported for the current scenario: Assign APIPortal.Administrator role to access the API portal APIs and perform operations like create, update, delete on various API portal entities as specified in the API Business Hub.

   ```json
   {   "role": "APIPortal.Administrator"
   }
   ```
Assign `APIPortal.Guest` role to access the API portal APIs in read-only mode. You can view the API portal entities as specified in the API Business Hub.

```json
{
  "role": "APIPortal.Guest"
}
```

Assign `APIManagement.SelfService.Administrator` role to configure additional virtual hosts.

```json
{
  "role": "APIManagement.SelfService.Administrator"
}
```

6. Click Next until you reach the Confirm section.
7. In the section Confirm, enter a unique Instance Name and choose Finish.

The creation of service instance is successful.

Now, with the help of the created service instance, generate a service key from the steps given below:

**Creating a Service Key**

1. Choose the created service instance link from the visible list.
2. In the left-hand pane, navigate to Service Keys → Create Service Key.
3. In the Create Service Key dialog that opens, provide a Name and Description (optional).
4. Click Save. The client credentials like `url`, `clientId`, `clientSecret`, and `tokenUrl` details appear for the given instance name.
   - The application `url` is used to make API calls.
   - The `clientId` and `clientSecret` are necessary credentials required to fetch the Bearer Token.
   - The `tokenUrl` is used to fetch the Bearer Token.
   - Example:

```json
{
  "url": "https://<apiportal application name>.cfapps.sap.hana.ondemand.com",
  "tokenUrl": "https://<Space name>.authentication.sap.hana.ondemand.com/oauth/token",
  "clientId": "sb-apiaccessxxxxxxxx!xxxx|api-portal-xsuaa!bxxxx",
  "clientSecret": "xxxxxxxxxxxxxxxxxxxxx="
}
```

Copy the client credentials in a notepad.

**Updating a Service Instance in the API Management, API portal**

You can update an already provisioned service instance of an API access plan by performing the following steps:

**Prerequisite:**
You must have the Cloud Foundry CLI installed.
1. Log in to the Cloud Foundry CLI by running the `cf login` command.
2. Select `Org`.
3. Run the following command to update your service instance. `cf update-service <service-instance-name> -c <empty-json-file>.json`.

### Sample Code

Sample json: `{}`

### Next Steps: Obtaining a Bearer Token

In the REST Console:

1. Paste the copied `tokenUrl`. Append `?grant_type=client_credentials` to the `tokenUrl`.
2. Choose Basic Auth as the Authorization type.
3. Similarly, paste the `clientId` and `clientSecret` in the place of Username and Password.
4. Make a POST Call.
5. Obtain the Bearer Token from the output and copy it in a notepad.

   - Now, to trigger an API, in the same REST Console, append the API endpoint (obtained from the API portal APIs that are located in the SAP API Management package of API Business Hub) to the `url`.

### Note

Currently, the `apiportal-apiaccess` plan allows you to access only the API portal APIs from the SAP API Management package.

- Choose Bearer Token as the Authorization type and paste the copied Bearer Token in the specified space.
- Include payloads, if needed.

### 1.4.1.4 API Management, API business hub enterprise as a Service

The API Management, API Business Hub Enterprise as a service on Cloud Foundry provides the API access plan.

- `apibhubenterprise-apiaccess`: The API business hub enterprise API Access plan allows you to generate a service key by creating a service instance. The service key, consisting of `url` (application url), `clientId`, `clientSecret`, and `tokenUrl` is used to generate a Bearer Token with the help of a REST Console. This Bearer Token, along with the application url and API endpoint are used to trigger the APIs. For more details, see API Access Plan for API business hub enterprise [page 59].
1.4.1.4.1 API Access Plan for API business hub enterprise

This topic explains how to enable API access for API Management, API business hub enterprise.

Prerequisites

- You have subscribed to the API business hub enterprise in the Cloud Foundry environment. For more information, see Setting Up API business hub enterprise Application [page 49].
- You have the space developer role assigned to you.
- You have created a service instance under the Authorization and Trust Management tile.
  1. In your web browser, open the SAP BTP Cockpit - https://account.hana.ondemand.com/cockpit.
  2. From your Subaccount, navigate to Spaces in your Cloud Foundry environment and choose Services Service Marketplace.
  4. In the Create Instance dialog that opens, choose the apiaccess plan.
  5. Click Next until you reach the Confirm section.
  6. In the section Confirm, enter a unique Instance Name and choose Finish.
- You have created a service key for the service instance above.
  1. Choose the service instance that you created above.
  2. In the left-hand pane, navigate to Service Keys Create Service Key.
  3. In the Create Service Key dialog that opens, provide a name.
  4. Click Save.
    The client credentials like url, clientId, and clientSecret details appear for the given service key.
- You have created a destination of type OAuth2Credentials to the XSUAA APIs by using the credentials you derived from creating the service key.
  1. From your Subaccount, navigate to Connectivity Destinations New Destination.
  2. Choose the service instance that you created above.
  3. In the Destination Configuration window, provide the details.

| i Note |
| You must enter the details exactly as mentioned below: |

Name: apimgmt-platform-access  
Type: HTTP  
Description:  
URL: https://yourxsuua.authentication.sap.hana.ondemand.com (Provide the value of the url field from the service key you created above.)  
Proxy Type: Internet  
Authentication: OAuth2ClientCredentials  
Client ID: apiaccess-client_id (Provide the value of the "clientid" field from the service key you created above.)  
Client Secret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxx (Provide the value of the "clientsecret" field from the service key you created above.)  
Token Service URL: https://yourxsuua.authentication.sap.hana.ondemand.com (Provide the value of the url field from the service key you created above.)  
Token Service User:
Token Service Password:

- For URL, provide the value of the `url` field from the service key you created above.
- For Client ID, provide the value of the `clientid` field from the service key you created above.
- For Client Secret, provide the value of the `clientsecret` field from the service key you created above.
- For the Token Service URL, provide the value of the `url` field from the service key you created above.

4. Click **Save**.

**About the Plan**

The API Access plan allows you to generate a service key by creating a service key. The service key, consisting of `url` (application url), `clientid`, `clientSecret`, and `tokenUrl` is used to generate a Bearer Token with the help of a REST Console. This Bearer Token, along with the application url and API endpoint, is used to trigger the APIs.

**Creating a Service Instance in the API Management, API business hub enterprise**

Create a service instance using API Access plan.

1. In your web browser, open the **SAP BTP Cockpit** - [https://account.hana.ondemand.com/cockpit](https://account.hana.ondemand.com/cockpit).
2. From your **Subaccount**, navigate to **Spaces** in your Cloud Foundry environment and choose **Services** > **Service Marketplace**.
3. Choose **API Management, API Business Hub Enterprise** > **Instances** > **New Instance**.
4. In the **Create Instance** dialog that opens, choose the plan as **devportal-apiaccess**.
5. Click **Next**.
6. In the section **Specify parameters**, provide the details as mentioned below, based on the role you require.

   - The roles that support API access in the API business hub enterprise are `AuthGroup.API.Admin` and `AuthGroup.API.ApplicationDeveloper`.

   **Create a service instance with the AuthGroup.API.Admin role to access the API business hub enterprise APIs (applications and attributes, API packages, API proxies and products, app developer and metering), and perform operations like create, update, and delete on various API business hub enterprise entities as specified in the API Business Hub.**

   ```json
   {
     "role": "AuthGroup.API.Admin"
   }
   ```

   **Create a service instance with the AuthGroup.API.ApplicationDeveloper role to access the API business hub enterprise APIs (applications, API packages, and API proxies and products), and perform operations like create, update, and delete on various API business hub enterprise entities as specified in the API Business Hub.**

   ```json
   {
   ```
i Note

What the developerId is:

Providing an invalid or an empty developerId throws an error in the service instance creation process.

To successfully create an application via the API business hub enterprise, you must provide a valid developerId. This means that you must have already registered as an application developer to the API Management, API business hub enterprise service or you must have been onboarded by your administrator.

- If you have registered to the API Management, API business hub enterprise application, provide your developerId.
  See the section below to know how to obtain your developerId.
- If you have not registered to the API Management, API business hub enterprise application, follow the steps in Register on API business hub enterprise [page 505] and try again.
- If you are not registered to the API Management, API business hub enterprise application, and require your admin to onboard you, contact your admin. See Onboard an Application Developer [page 504].

How to obtain the developerId:

- If you are a registered developer in the API business hub enterprise, access the following URL in your browser to obtain your developerId:

  ```
  https://devportal-url/api/1.0/user
  #Response
  [{"Name":"
  "FirstName":"
  "LastName":"
  "LoggedOut":false,
  "Email":""}]
  ```

  The Name field in the response is your developerId.

- If you are an admin and are obtaining the developerId for a developer you have already onboarded, pick the userId that you provided during the developer onboarding.

To view a list of the registered developers, access the following URL in your browser. The userId field in the response is the developerId.

  ```
  https://devportal-url/api/1.0/registrations?type=registered
  #Response
  autoReLogin: false
  country: ""
  emailId: ""
  firstName: ""
  lastName: ""
  rolesAccess: [{status: "registered", role: "API_ApplicationDeveloper"}]
  0: {status: "registered", role: "API_ApplicationDeveloper"}
  userId: ""
  ```

Limitation: Self-service onboarding request is not supported for a developer. So, the POST operation under the API Business Hub Enterprise - Registering Users tile in the API Business Hub cannot be
made by the application developer service key. As an alternative, you can invoke this API using the admin service key.

7. In the section Confirm, enter a unique Instance Name, and choose Finish.

The service instance is successfully created and listed in the Instances window.

Create a Service Key

Generate a service key for the service instance that you created above:

1. From the Instances window, choose the service instance that you created above.
2. In the left-hand pane, navigate to Service Keys ➤ Create Service Key ➤ Create Service Key.
3. In the Create Service Key dialog that opens, provide a name.
4. Click Save.

The credentials like url, tokenUrl, developerId (for developer role), clientId, and clientSecret details are displayed for the given service key.

- The application url is used to make API calls.
- The clientId and clientSecret are necessary credentials required to fetch the Bearer Token.
- The tokenUrl is used to fetch the Bearer Token.

Sample Code

Example (for admin role)

```json
{
  "url": "https://developer-portal-application-url",
  "tokenUrl": "https://token-endpoint-url/oauth/token",
  "clientId": "your-admin-client-id",
  "clientSecret": "xxxxxxxxxxxxxxxxxxxxx="
}
```

Example (for developer role)

```json
{
  "url": "https://developer-portal-application-url",
  "tokenUrl": "https://token-endpoint-url/oauth/token",
  "developerId": "developerID-associated-with-the-current-instance",
  "clientId": "your-dev-client-id",
  "clientSecret": "xxxxxxxxxxxxxxxxxxxxx="
}
```

Note these credentials to use them in the next steps to obtain a bearer token, in order to access the API business hub enterprise APIs.
Updating a Service Instance in the API Management, API business hub enterprise

You can update an already provisioned service instance of an API access plan by performing the following steps:

Prerequisite:
You must have the Cloud Foundry CLI installed.

1. Log in to the Cloud Foundry CLI by running the `cf login` command.
2. Select Org.
3. Run the following command to update your service instance. `cf update-service <service-instance-name> -c <empty-json-file>.json`.

```shell
Sample Code
Sample json: {}
```

Next Steps

Obtaining a Bearer Token

In the REST Console:

1. Paste the copied `tokenUrl`. Append `?grant_type=client_credentials` to the `tokenUrl`.
2. Choose Basic Auth as the Authorization type.
3. Similarly, paste the `clientId` and `clientSecret` in the place of Username and Password.
4. Make a POST Call.
5. Obtain the Bearer Token from the output and copy it in a notepad.
   - Now, to trigger an API, in the same REST Console, append the API endpoint (obtained from the API business hub enterprise APIs that are located in the SAP API Management package of API Business Hub) to the `url`.
   - Choose Bearer Token as the Authorization type and paste the copied Bearer Token in the specified space.
   - Include payloads, if needed.
   - Make an API call.

1.4.1.5 Account Members

All members in your enterprise who need to use API Management applications need to be assigned to the account.

Members can use the application within the scope of the account and based on the roles assigned to the members. For information on how to assign members to the account, see Managing Members.
1.4.1.6 Assigning User Roles

Use role collections to group together different roles that can be assigned to API Portal and API business hub enterprise users.

As an administrator of SAP API Management in the Cloud Foundry environment, you can maintain API Portal and API business hub enterprise roles and role collections, which can be used in user management. Typically, a role collection consists of one or multiple roles. You assign roles to role collections, which are in turn assigned to users. Using the SAP BTP cockpit, you can display information about the role collections that have been maintained as well as the roles available in a role collection. For more information, see Roles and Role Collections.

The following predefined roles are shipped with API Portal. These roles by default, are shared, visible, and accessible within all the accounts that have subscribed to API Portal:

<table>
<thead>
<tr>
<th>Role Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIPortal.Administrator</td>
<td>Use this role to access the API portal user interface (UI) and services, manage the API proxies by adding additional policies. You can also use this role to manage APIs using the API Designer.</td>
</tr>
<tr>
<td>APIPortal.Service.CatalogIntegration</td>
<td>Use this role to establish a connection from the API business hub enterprise to the API portal.</td>
</tr>
<tr>
<td>APIManagement.Selfservice.Administrator</td>
<td>Use this role during the onboarding of API Portal and to get access to its settings page.</td>
</tr>
<tr>
<td>APIPortal.Guest</td>
<td>Use this role to access the API portal in read-only mode. You can view all APIs, policies, API providers, and analytics, but can’t edit them.</td>
</tr>
</tbody>
</table>

The following predefined roles are shipped with API business hub enterprise:

<table>
<thead>
<tr>
<th>Role Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthGroup.SelfService.Admin</td>
<td>Use this role during the onboarding of API business hub enterprise and to get access to it.</td>
</tr>
<tr>
<td>Role Collection</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>AuthGroup.API.Admin</td>
<td>Use this role to:</td>
</tr>
<tr>
<td></td>
<td>- Manage an application developer’s access to the portal by either accepting or rejecting an application developer’s request. In addition, you can revoke the access of an existing application developer.</td>
</tr>
<tr>
<td></td>
<td>- Manage roles for a user by adding new roles or removing existing roles.</td>
</tr>
<tr>
<td></td>
<td>- On-behalf of an application developer, admin can also perform the following tasks:</td>
</tr>
<tr>
<td></td>
<td>○ Create, update, and delete applications.</td>
</tr>
<tr>
<td></td>
<td>○ Create custom attributes for applications.</td>
</tr>
<tr>
<td></td>
<td>○ Provide app key and secret, while creating or updating an application.</td>
</tr>
<tr>
<td>AuthGroup.ContentAuthor</td>
<td>Use this role to:</td>
</tr>
<tr>
<td></td>
<td>- Publish content to the API business hub enterprise.</td>
</tr>
<tr>
<td></td>
<td>- Establish a connection from the API portal to the API business hub enterprise.</td>
</tr>
<tr>
<td>Role Collection</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| AuthGroup.API.ApplicationDeveloper | Use this role to:  
  - Access the API business hub enterprise.  
  - Create, update, and delete applications.  
  - View analytics information on application usage, performance, and error count.  
  - View and download bills for subscribed applications.  

**Note**  
The AuthGroup.API.ApplicationDeveloper role must not be assigned manually to a user form the SAP BTP Cockpit and this role must not be a part of any user group assignment.  
This role is assigned by default to a user who onboards to the API business hub enterprise using the Self-registration process or via Add User flow.  
A user must be onboared to API business hub enterprise only via Self-registration or Add User flow. For more information on registering in API business hub enterprise, see Register on API business hub enterprise [page 505]. In the Add User flow, the API business hub enterprise admin adds a user who wants to be onboarded to API business hub enterprise. However, the user who is requesting to be onboarded must ensure that the user details provided to the admin matches the user details obtained from the response of <developer portal url>/api/1.0/users. |
| AuthGroup.Content.Admin       | Use this role to:  
  - Create and update categories.  |
| AuthGroup.Site.Admin           | Use this role to:  
  - Configure updates.  
  - Perform portal changes like uploading logo, changing the name and description, and changing the footer links. |

For more information, see Assign Role Collections and Assigning Role Collections to Users [page 67]
1.4.1.6.1 Assigning Role Collections to Users

Role collections enable you to group together the roles you create. The role collections you define can be assigned to users logged on to SAP ID service.

Procedure

1. In your web browser, open SAP BTP Cockpit and choose the relevant subaccount.
2. In the left-hand pane, choose Security > Role Collections.
3. Choose the role collection to which you want to assign users.
4. Go to the Users section and choose Edit.
5. Enter the user ID of the user that you want to assign to the role collection. If the user only exists in a connected identity provider, you must choose the identity provider and type in the e-mail address.
6. (Optional) To add more users, choose Add a user.
7. Save your changes.

Related Information

Setting Up API Portal Application [page 43]
Setting Up API business hub enterprise Application [page 49]
Creation of Shadow Users [page 72]

1.4.1.6.2 Creating a Custom Role

Create a custom role for API Products in API Management.

Context

You can restrict access to an API product in API Management using a custom role. That is, only an authorized user can discover and subscribe to API Products that are tagged to a custom role.

Important

If you’re using Integration Suite, refer Create Roles for Applications Using Existing Role Templates to create a custom role for API Products.

To create a custom role for API Product, use the ApplicationDeveloper role template. Also, ensure that for the CustomRole attribute, you choose the value of Source as Static, and in the Values, specify the attribute values and press enter. This value is later used to assign permission while creating an API Product.
Procedure

1. Go to your Subaccount in SAP BTP Cockpit for Cloud Foundry environment.
2. Choose Service Marketplace in the left-hand pane.
3. In order to create a custom role, choose the API Management, API Business Hub Enterprise tile.
4. Under Application Plans for API BusinessHub Enterprise, choose the **Action** icon and select Manage Roles.
5. To add a new custom role, choose + Add a role.
6. In the Create Role dialog, fill the details for:
   - Role Name
   - Description
   - Role Template: Choose ApplicationDeveloper.
   - Attributes: For the CustomRole attribute, keep the value of Source as static. Under Values specify the attribute values and press enter. This value is later used to assign permission while creating an API Product.

A new role is created and added to the Roles list.

7. **Add the created role to Role Collection:** Adding a custom role to the role collection ensures that you choose a specific application and role template.
   a. Navigate to your Subaccount > Security > Role Collections and choose + Create New Role Collection and provide a Name and Description to the new role collection.
   b. Choose the newly created Role Collection and choose Edit.
   c. To select the custom role, choose the icon under the Roles tab.
   d. On the Select: Role dialog, choose the custom role from the Role Name dropdown, select the checkbox under Roles, and choose Add.
   e. Choose Save.
8. **Assign role collection to the user:** To assign the created role collection to your authorized email ID:
   1. Go to the Users section and choose Edit.
   2. Enter the user ID of the user that you want to assign to the role collection. If the user only exists in a connected identity provider, you must choose the identity provider and type in the email address.
   3. (Optional) To add more users, choose + (Add a user).
   4. Save your changes.

> **Remember**
Application Developers who are already onboarded in the API business hub enterprise should have the custom role. If any user has been assigned a custom role but hasn’t been onboarded as an application developer in the API business hub enterprise, the application creation fails. In this case, Authgroup.API.Admin can onboard the user as an Application Developer in the portal.

**Next Steps**
After completing the above steps, assign permissions while creating a Product in API Portal application. For more information on the same refer, here [page 487].
1.4.2 Custom Domain Configuration for API Portal or API Business Hub Enterprise Subscription

To complete the process of configuring a custom domain for the API Portal or the API business hub enterprise application using the Custom Domain Service in the SAP BTP Cloud Foundry environment, you need to contact the SAP API Management operations team.

**i Note**

Custom domain for subscription URLs is available for API Management, API portal, API Management, API business hub enterprise standard plan, and API Management as a capability within the Integration Suite product.

For information on the initial setup of the Custom Domain Service in the Cloud Foundry environment, see Configuring Custom Domains.

To map the custom domain to a SAAS application subscription, see Map a SaaS Application to a Custom Domain.

After mapping a custom domain to a SAAS application subscription using the Custom Domain Service, you need to raise a ticket through the SAP Support Portal to complete the SaaS route configuration.

Use the following component for your incident:

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPU-API-OD-OPS</td>
<td>SAP API Management Operations - On Demand</td>
</tr>
</tbody>
</table>

When submitting the incident, include the following information:

- Subdomain of the current account
- Application: API Portal/ API business hub enterprise
- Custom domain URL (the host name for the URL)
- API Portal/API business hub enterprise URL
1.4.2.1 Requesting for Custom Domain for a Virtual Host

When you subscribe to API Management service, and are creating API Proxies, they get created with the default virtual host, and the default domain. To suit your requirements, you can request for a custom domain.

Context

You can secure the custom domain with either a one-way or a two-way SSL certificate. SSL is a standard security protocol that encrypts the connection between a web server and a web client, for example the connection between a web browser and an application. Here, the one-way SSL validates only the identity of the web-server but the two-way SSL validates the identities of both the web server and the web client. For requesting the two-way SSL certificate, see Requesting for Two-Way SSL Certificate [page 71].

To request a custom domain with one-way SSL, perform the following steps:

Procedure

1. Create an incident using the link https://launchpad.support.sap.com/
2. In the Create Incident window, expand the Description category and enter the following details in the description field:
   a. The custom domain name that you want.
   b. Subdomain Name and ID: You find the subdomain name and ID by opening SAP BTP Cockpit <Your_subaccount> On the Overview page of your subaccount, make a note of the value marked against the Subdomain attribute as well as the ID.
   c. An e-mail ID that you intend to use for sharing the secure drive link. The secure drive link is provided once you raise this incident, and you use the secure drive link to upload private key, public certificate, and the relevant chain certificate.
   d. One-way SSL authentication: Enabled
   e. Component: OPU-API-OD-OPS
3. Submit the incident.

   i Note

   It’s recommended that you redeploy the API Proxies that are linked to the custom domain virtual host, after the virtual host is updated.

Related Information

Requesting an Additional Virtual Host in Cloud Foundry Environment [page 46]
1.4.3 Requesting for Two-Way SSL Certificate

Request a two-way SSL certificate for the default domain of the virtual host of your API Management service.

**i Note**
The default domain of the virtual host provided with your API Management service is secured with a one-way SSL certificate. For more information, see Requesting for Custom Domain for a Virtual Host [page 70].

You can request for a two-way SSL certificate, which validates the identities of both the web server and the web client.

**Procedure**

1. Create an incident using the link https://launchpad.support.sap.com/
2. In the Create Incident window, expand the Description category and enter the following details:
   1. The virtual host alias. For the default domain, this is the default virtual host alias.
   2. Subdomain Name and ID: Go to SAP BTP Cockpit <Your_subaccount> On the Overview page of your subaccount, make a note of the value marked against the Subdomain attribute as well as the ID.
   3. Two-way SSL authentication: Enabled
   4. An e-mail ID that you intend to use for sharing the secure drive link. The secure drive link is provided once you raise this incident, and you use the secure drive link to upload the public certificate, and the relevant chain certificate.
   5. Component: OPU-API-OD-OPS
3. Submit the incident.

1.4.4 Requesting Egress NAT IP Addresses for API Management Service in Different Regions

API Management protects your backend services. However, API Management needs to establish connectivity to your backend services during an API call execution.

In case your backend service is restricting access to certain IPs as part of security measures, you need to add API Management NAT IPs to the list of allowed IPs in your backend services.

To get region-specific egress NAT IP addresses for API Management, raise a ticket through the SAP Support Portal. For more information, see Product Support.

Use the following component for your incident:

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPU-API-OD-OPS</td>
<td>SAP API Management Operations - On Demand</td>
</tr>
</tbody>
</table>
1.4.5 Creation of Shadow Users

Whenever a user authenticates at an application in your subaccount using any identity provider, it’s essential that user-related data provided by the identity provider is stored in the form of shadow users.

Previously the User Account and Authentication service allowed any user of any connected identity provider to authenticate to applications in the subaccount. When there was no corresponding shadow user, it automatically created one based on the information received from the identity provider.

With new subaccounts created after 24 September 2020, automatic creation of shadow users is switched off by default for the default identity provider, SAP ID service. You can enable or disable automatic shadow user creation using the information here.

Since automatic shadow user creation is disabled, if you’ve not explicitly created shadow users for your developers, then they’re unable to log on to the application, and they’re asked to contact the administrator.

You can create the shadow users for your developers in the SAP BTP cockpit, typically when you assign the first role collection to them. For more information on role collection assignment, see Assigning Role Collections to Users or User Groups.

For information on creating shadow accounts, see Add Users to SAP ID Service in the Cloud Foundry Environment.

You can also use the User Management (System for Cross-domain Identity Management (SCIM)) API to manage your shadow users.

1.4.6 Cancelling API Management Service Subscription

Cancel your API Portal application and API business hub enterprise application subscriptions to disable your account from the API Management service.

Context

→ Remember

If you’re using standalone API Management, ensure you first cancel the subscription for the API business hub enterprise application, followed by the API Portal application.

If you’re using API Management via the Integration Suite, you can unsubscribe from the applications in any order of your preference.

Unsubscribeing deletes the data stored in the respective applications.
Procedure

1. Go to [SAP BTP Cockpit ➤ Subaccount] on Cloud Foundry environment.
2. Choose Subscriptions on the left-hand pane.
3. Choose the API Business Hub Enterprise tile.
4. Select Unsubscribe.
5. Navigate to your Subaccount and choose the API Management, API portal tile.
6. Choose Unsubscribe.

1.4.7 Centralized API business hub enterprise

The centralized API business hub enterprise is a central API catalog, allowing application developers to consume APIs and other assets, from a common platform.

Example

Organizations can use the centralized API business hub enterprise as a single platform, where different lines of business publish their APIs, allowing the organization’s consumers to access these varied APIs from one location.

Every API Management instance will have an API business hub enterprise application on cloud. One of the various API Business Hub Enterprises is set as a centralized API catalog. This catalog will then receive API proxies, API products, and other assets from each connected API Portal. All the assets published to the centralized API business hub enterprise must be unique.

Remember

You can define different API Management instances to suit a particular stage in the API lifecycle. So you can have Development, Test, and Productive instances. Transport of APIs between the API Portals of these instances is also possible, see Transport APIs and Its Related Artifacts [page 534]. However, connecting the API Portals having such a relationship to the same API business hub enterprise will violate the uniqueness of the assets.

Once the application developers register to centralized API business hub enterprise, they can seamlessly search, explore, and test APIs. They can also create and subscribe to various applications from the API business hub enterprise.

The API business hub enterprise admin identifies which existing or new API Business Hub Enterprise application can accept content from multiple API portals.

Creating a Connection Request for the Centralized API business hub enterprise [page 74]
To publish the API portal content on the API business hub enterprise, create a request to connect the API portal to the API business hub enterprise.

Approving Pending Connection Requests [page 77]
As an API Business Hub Enterprise administrator, you must approve or reject the connection request after you receive them.
1.4.71 Creating a Connection Request for the Centralized API business hub enterprise

To publish the API portal content on the API business hub enterprise, create a request to connect the API portal to the API business hub enterprise.

Prerequisites

- Raise an offline request to the API business hub enterprise administrator for assigning the `AuthGroup.APIPortalRegistration` role to you.

  **Note**
  
  You can't create a connection request, without the `AuthGroup.APIPortalRegistration` role.

- Generate the access credentials to establish the connection. To generate the credentials from the API Portal, you must have the `APIPortal.Administrator` role assigned to you.
  1. Log in to the API portal.
  2. Navigate to Onboarding Settings and choose Connection.
  3. Follow the onscreen instructions under Connect the API Portal to the centralized API Business Hub Enterprise to generate the API Portal access credentials.

  **Note**
  
  The client credentials get generated for the `APIPortal.Service.CatalogIntegration` role.
**Context**

The API business hub enterprise administrator identifies which existing or new API business hub enterprise application can accept content from multiple API portals.

**i Note**

Only new API Management subscriptions are allowed to set up a connection with the API Business Hub Enterprise.

**i Note**

You can connect a maximum number of three API portals to the centralized API business hub enterprise.

Create a new subaccount in Cloud Foundry and set up only the API portal. See [Setting Up API Portal Application](#) to subscribe to API Management, API portal in the Cloud Foundry.

For the newly set up API portal, you can request for the API Business Hub Enterprise connection to be established.

**i Note**

If an API portal is already connected to an API business hub enterprise, sever the connection between the existing API business hub enterprise to set up a new connection with the centralized API business hub enterprise. The option to disable or severe the connection with an existing API business hub enterprise from isn’t supported currently.

**Procedure**

1. Log on to the *API Business Hub Enterprise*.
2. Navigate to the *Manage* tab and choose *Establish API Portal Connectivity with API Business Hub Enterprise* tile.
3. Choose ‌ icon to create a request to connect the API portal to the centralized API business hub enterprise.

**i Note**

Once this connection is set up, you can’t place a request to severe this connection and establish a new connection with any other API business hub enterprise.

Choose *OK* on the *Confirmation* screen to proceed.
4. Fill in the following details on the *Submit Connection Request* page.
Parameters | Values
--- | ---
API Portal Alias Name | Enter the API portal name that gets displayed on the API Business Hub Enterprise. This name is used to distinguish products that are published from the API portal and likewise for applications created for the product.

API Portal Access Credentials | Enter the API portal access credentials that you generated earlier. These credentials are used by the API Business Hub Enterprise to establish the connection.

Comment | Provide the details to the approver about the need for the connection request.

5. Choose Submit.

Results

You’ve submitted the connection request to the API business hub enterprise administrator. Once the connection request is approved by the administrator, you can start publishing the API portal content to the API business hub enterprise.

Task overview: Centralized API business hub enterprise [page 73]

Related Information

Approving Pending Connection Requests [page 77]

1.4.7.1.1 Updating the Connection Request Credentials

There can be instances where you have to update the credentials you’ve used to establish a connection between the API portal and the API Business Hub Enterprise.

Context

Update the API Portal Access Credentials when you encounter one of the following situations:

- You have submitted incorrect credentials while raising a connection request, and your request is in pending approval or submitted state.
You've deleted the service instance, or the service key, after the connection between the API portal and the API Business Hub Enterprise was established. In this case, the credentials you were using before deleting the service instance or the service key becomes invalid.

**Procedure**

1. Log on to the *API Business Hub Enterprise*.
2. Navigate to the Manage tab and choose *Establish API Portal Connectivity with API Business Hub Enterprise* tile.
3. Go to the Actions column of the connection request and choose *Edit Credentials*.
4. Enter the Client ID and Client Secret and choose Save.

**Results**

You’ve updated the API Portal access credentials successfully.

### 1.4.7.2 Approving Pending Connection Requests

As an API Business Hub Enterprise administrator, you must approve or reject the connection request after you receive them.

**Prerequisites**

You’re assigned the AuthGroup.API.Admin role.

**Context**

You can access the connection requests pending for approval from the Manage API Connections tile.

**Procedure**

1. Log on to the *API Business Hub Enterprise*.
2. Navigate to the Manage tab and choose Manage API Portal Connections tile.
The connection requests that are pending for approval are listed on the Manage API Portal Connections page.

3. Choose View to read the comments from the requester before approving or rejecting a connection request.

4. Choose Manage Connections icon in the Actions column and choose Approve.

**Results**

The connection has been set up between the API portal and the API Business Hub Enterprise.

**Task overview:** Centralized API business hub enterprise [page 73]

**Related Information**

Creating a Connection Request for the Centralized API business hub enterprise [page 74]

1.4.7.2.1 Updating the Connection Request Credentials for an Approved Request

There can be instances where you have to update the credentials once the connection request is approved by the API business hub enterprise admin.

**Prerequisites**

To update the API portal access credentials, you must first generate it. To generate the credentials from the API Portal, you must have the APIPortal Administrator role assigned to you.

1. Log in to the API portal.
2. Navigate to Onboarding Settings and choose the Connection tab.
3. Choose Regenerate Credentials and Copy the API Portal access credentials.

**i Note**

The client credentials get generated for APIPortal.Service.CatalogIntegration role.
**Context**

To establish the connection between the API portal and the API business hub enterprise, the client id and client secret created for the API Portal is shared during the connection request process.

If you encounter one of the following situations after the connection request has already been approved by the API business hub enterprise admin, you have to update the credentials:

- The service instance, or the service key gets deleted after the connection between the API portal and the API business hub enterprise was established. In this case, the credentials you were using before the service instance or the service key got deleted becomes invalid.
- Similarly, if the destination that fetches the API content from the API Portal workspace gets deleted, the credentials you were using before the destination got deleted becomes invalid.

**Procedure**

1. Log on to the *API Business Hub Enterprise*.
2. Navigate to the Manage tab and choose *Establish API Portal Connectivity with API Business Hub Enterprise* tile.
3. Go to the Actions column and select the approved connection request that you want to edit and choose Re-establish Connection.
4. On the *Submit Connection Request* page, enter the *Client ID* and *Client Secret* that you copied earlier from the API Portal.
5. Choose *Save*.

**Results**

You’ve updated the API Portal access credentials successfully.

**1.5 Development**

API Management provides **one experience** for managing and monitoring all APIs across various data platforms and is enriched with real-time analytics. API Management enables consumers to access relevant data directly in a secure manner. Selective data can be exposed while reducing the risk of security breaches. Rather than app developers consuming services directly, they access APIs created using API Management. APIs created using API Management map a publicly available HTTP endpoint to backend services. API Management handles the security and authorizations required to protect, analyze, and monitor your services.
The following table lists the tasks that an Application Developer, Application Consumer and an API Administrator can perform in SAP API Management:

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Applicable to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Setup [page 42]</td>
<td>Regardless of your role, familiarize yourself with core API Management concepts. As an administrator, learn about the activities required before APIs can be built using API Management.</td>
<td></td>
</tr>
<tr>
<td>Build APIs</td>
<td><strong>API Platform:</strong> To expose an API, you first need to create an API Provider. You can then create an API proxy and associate it with policies.</td>
<td>API Administrator</td>
</tr>
<tr>
<td>Publish APIs</td>
<td><strong>API Platform:</strong> To publish your APIs, you need to create a product.</td>
<td>API Administrator</td>
</tr>
<tr>
<td>Analyze APIs</td>
<td><strong>API Analytics:</strong> Use the capabilities of API Analytics to analyze API usage, performance, and potential errors.</td>
<td>API Administrator</td>
</tr>
<tr>
<td>Consume APIs</td>
<td><strong>Developer Services:</strong> Use the API Management API business hub enterprise to consume the exposed APIs.</td>
<td>Application Developer and Application Consumer</td>
</tr>
</tbody>
</table>

### 1.5.1 Build APIs

API portal provides a common platform for API designers to define and publish APIs. Every API Management customer is provided with their own API portal application on cloud. The API portal offers capabilities to configure systems, build and publish APIs, analyze and test APIs.

### Prerequisites

Before you start the process of building APIs, it is important to understand the different artifacts associated to an API. For more information, see API Artifacts [page 84].
To expose an API, you first need to create a system so you can connect to the API provider. After you have done this, you can create APIs by associating policies to it. Once you associate the policies and your API is ready to use, you test it using the API test console.

To build an API, you need to perform the following tasks:

### Procedure

1. Create an API Provider [page 374].
2. Create an API Proxy [page 390].
3. Associate policies to an API [page 92].
4. Test APIs using the API Test Console [page 464].

### Note

In order to achieve an effortless navigation to the API business hub enterprise, choose Navigation Links ( bölümü) from the API portal and select *API Business Hub Enterprise*. 
1.5.1.1 OpenAPI Specification 3.0

In addition to supporting OAS 2.0, API Management now supports OpenAPI Specification (OAS) 3.0. API Management supports creation and import of API definitions in Open API Specification (OAS) 3.0. These OAS 3.0 API definitions can be created using API designer or existing ones can be imported using API Management import functionality. For the created or imported API definition, API proxy is created on which user can apply policies to bring in API Management capabilities.

To know more about importing APIs, see Import an API [page 448].
To know more about creating APIs using the API designer, see Create an API from API Designer [page 405].

Limitations

- External references aren’t supported with OAS 3.0 (or OAS 2.0) in API Management.
- Images aren’t supported with OAS 3.0 in API Management.
- Local and remote links aren’t supported with OAS 3.0 API Management.
- API proxy creation is not possible by discovering the OAS 3.0 APIs from SAP API Business Hub. This feature will be enabled soon.

1.5.1.2 Externally Managed APIs

You can manage APIs that are not managed by SAP API Management but by some other API Management solution.

You can import an API definition (Open API Specification 2.0 or 3.0) of externally managed APIs in API portal, without adding management capability (proxy creation and policies) of SAP API Management and publish them to API business hub enterprise.

Features of Externally Managed APIs

- These APIs may be managed by external gateways.
- API Proxies are not created for these APIs.
- These APIs are only listed in SAP API Management. No aspect of their life cycle is managed by API Management.
- Status (Deployed/Not Deployed) is not displayed for these APIs.
- These APIs are represented by the symbol: 🔐
- If you create a product using only externally managed APIs, your consumers can’t view the rate plan for these APIs, nor can they subscribe to the product or use custom attributes for the product in the API business hub enterprise. For more information, see Create a Product [page 482].
• You can choose to manage the life cycle of these externally managed APIs after listing them. To do so, from the Overview page of the API, choose Manage.
• You can list externally managed APIs by importing them. For more information, see Import an API [page 448].

1.5.1.2.1 Converting Externally Managed APIs to Internally Managed APIs

You can convert an external API, whose lifecycle is managed by an external API Management solution, to an internal API, which is managed by SAP API Management.

Context

After the conversion, you can create API proxies for these internally managed APIs, import, and publish them and apply policies to them. You can apply all the SAP API Management capabilities to these managed APIs.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. Identify the externally managed API that you want to convert from the list of APIs.

   i Note
   The externally managed API is marked with an icon, and the Status column doesn’t display the status of the API.

4. Choose the Action icon against the required API and then select the Manage option. Alternatively, you can open the required API, and then select the option Manage.
5. You can edit the prefilled data of the externally managed API before choosing Deploy.

   i Note
   You can attach policies to the API only after it’s deployed.

6. To attach policies, navigate back to the list of APIs on the Develop page, choose the externally managed API that you’ve converted into an internally managed API.

   You can see that the status column now shows the status of the API as Deployed.
7. Open the required API, and choose Policies on the details screen. See Create a Policy [page 451] for more information.
Results

You’ve converted an externally managed API into an internally managed API and have applied policies to that managed API.

Next Steps

To import and publish the internally managed APIs, refer the following topics: Import an API [page 448] and Publish APIs [page 481]

1.5.1.3 API Artifacts

Describes the various entities of an API.

This section introduces the various artifacts of an API that you need to know before building APIs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Proxy</td>
<td>Is a discrete representation of an API entity that abstracts the actual proxy end point properties at one end and the actual target endpoint (the endpoint that is relevant for the end user to invoke) at the other end. It also includes other properties that describe the policies that need to be invoked on the API, the attachments, and documents, and other artifacts that are relevant to the API.</td>
</tr>
<tr>
<td>Proxy Endpoint</td>
<td>Manages interactions with API consumers. Consumers of the API normally interact with the base path of the API and are attached to policy entities that operate to define quota, access limiters, and so on.</td>
</tr>
<tr>
<td>Target Endpoint</td>
<td>Manages interactions with the backend service endpoint on behalf of consumer applications. Backend endpoint forwards request messages to the proper backend service.</td>
</tr>
<tr>
<td>API Resource</td>
<td>Individual business entities that an API proxy contains. For example: BusinessPartnerCollection is an API resource that the API administrator would like to present via an API Proxy entity.</td>
</tr>
<tr>
<td>Operations</td>
<td>Is the object representation to specify if GET, POST, PUT, and DELETE calls are specified.</td>
</tr>
<tr>
<td>Policy</td>
<td>The runtime engine of SAP API Management is policy driven. This means that policies are decoupled from the service definition. They can be dynamically linked to these APIs or services to enforce minimal or maximum levels of operation and Quality of Service.</td>
</tr>
<tr>
<td>API Documentation</td>
<td>Describes each API resource in a simple and concise manner.</td>
</tr>
</tbody>
</table>
1.5.1.3.1  API Proxy

An API is exposed in API Management as an API proxy. An API proxy is a discrete representation of an API. It is implemented as a set of configuration files, policies, and code snippets that rely on the resource information provided by API Management.

The API proxy decouples an API from any backend changes. This provides flexibility to application developers to continue calling the same API.

Supported Service Types

Broadly API Proxies can be exposed as REST, ODATA, and SOAP APIs. For example, a backend RESTful service can be exposed directly as REST AP. An ODATA service can be exposed either as an ODATA API or even a REST API. A SOAP service can be exposed as a pass-through SOAP API directly. The benefit of exposing a service as an ODATA API is that the exposed API will comply with ODATA-specific operations (like metadata fetch, navigating through associations and so on). You have the flexibility of exposing an ODATA service also as a RESTful API. But in doing so, you also need to ensure that the REST resource is mapped correctly to the ODATA resource. When you expose a SOAP service as a SOAP API, there is no strict notion of an API resource as SOAP services work directly on the endpoint. Every operation-type on the SOAP service is as per the WSDL contract and does not directly map to the exposed resource.

API proxies handle request and response messages as a processing pipeline. In an API proxy configuration, there are two types of endpoints: Proxy Endpoint and Target Endpoint.

Proxy Endpoint

The proxy endpoint defines the settings for the inbound connections for an API proxy. When you configure a proxy endpoint, you define how the client applications should invoke the API proxy. The main purpose of this configuration object is to manage interactions with consumers of the API. An API proxy must contain a proxy endpoint.

Target Endpoint

The Target endpoint defines the outbound connections for an API proxy. The main purpose of this object is to manage interactions to the actual backend service endpoint on behalf of consumer applications. An API proxy can contain zero or many target endpoints.

1.5.1.3.2  Flow

A Flow defines a processing pipeline which controls how the API behaves and defines what information it should carry.

A processing pipeline comprises of a Request and a Response stream. Proxy endpoint and target endpoint define a pipeline to process request and response messages. A flow is a request or response processing
pipeline defined by a proxy endpoint or target endpoint. Each request or response flow is subdivided into a PreFlow, one or more optional Conditional Flow, a Post Flow, and an optional PostClient Flow.

- **PreFlow**: This flow is always executed as the first step in the segment where it is applied, before the conditional flow. Configure a PreFlow when you want to ensure a policy is executed before anything else. Use the PreFlow on the proxy endpoint for example, when you don’t want a call that has exceeded its quota to be routed to the backend layer, or when you have to authenticate users. To support such requirements, you usually put quota and security policies in the PreFlow pipeline. This ensures that the policies will always execute before any other processing takes place.

- **Conditional Flow**: A condition associated to a flow. A flow can contain one or more conditions. However, only the first condition met is executed. Configure a conditional Flow when you want a set of policies to be executed only when a condition is met. You can define multiple conditional Flows. However, a conditional flow segment is executed only when a match is found with the criteria defined in the Conditional String. Once a conditional Flow is executed, all other succeeding conditional Flows along the chain will not be executed. For example, you want to convert XML to JSON only when the requesting application is running on a mobile device. This scenario can be configured by setting up conditional Flows.

- **PostFlow**: This flow is always executed as the last step in the segment where it is applied, after the conditional flow. Configure a PostFlow when you want to log some data or send a notification that something happened. A PostFlow is always executed regardless of the situation.

- **PostClientFlow**: This is an optional flow that executes after the response message has been sent to the requesting client application. You can add a PostClientFlow only to the response flow of a ProxyEndpoint. PostClientFlow reduces API proxy latency and makes information available for logging that is not calculated until after the response is returned to the client.

**Note**

PostClientFlow is executable via the import functionality, it is executed only when you import an API proxy that contains the PostClientFlow standard payload. The sample payload is provided below for your reference. You can attach only Message Logging policies to a PostClientFlow.

To execute a PostClientFlow, perform the following:

1. Export the required API proxy from API portal. For more information, see Export an API [page 450].
2. Add the below sample payload starting from the line `<postClientFlow>` in the default.xml file available under APIProxyEndPoint folder of your API proxy. For more information, see API Proxy Structure [page 352].

**Sample Code**

```
<postFlow>........
       ...........</postFlow>
    <postClientFlow><name>PostClientFlow</name>
    <response>
    <steps>
    <step>
     <policy_name>clientflowMessagePolicy</policy_name>
     <condition> </condition>
     <sequence>1</sequence>
    </step>
    </steps>
    </response>
</postClientFlow>
```
In the payload, ensure that the policy name entered in the `<policy_name>` field is an existing policy that belongs to your API proxy. The Policy folder displays all the policies that are currently attached to your API proxy.

3. Import the updated API proxy in API portal. For more information, see Import an API [page 448]

A policy can be assigned to any of the above four flow types. You configure a PreFlow and PostFlow in the proxy endpoint or target endpoint configurations only when you want to enforce a policy.

**Defining Flows in Policy Designer**

Use the policy designer to define Flows and policies. The policy designer allows you to define one PreFlow, one PostFlow and zero or more Conditional Flows on the proxy endpoint and target endpoint individually. You can also choose to have no conditional Flows on the proxy endpoint or target endpoint.

You can assign one or more policies to each PreFlow, PostFlow or Conditional Flow. The list of supported policies is available on the right under the Policies section. The count of the policies attached to a Flow is depicted as a number beside the Flow. To view the list of policies attached to a flow, for example on a PreFlow, select PreFlow under proxy endpoint in the Flows section. The Policy designer will visually display all policies attached on this PreFlow for the proxy endpoint. On selecting a policy, you can view the details of the Conditional String as well as the content of the Policy itself. The Policy is executed only if the conditional String element on the Policy evaluates to true. You can similarly attach policies to a PostFlow or Conditional Flow.

You enter the conditions in the Conditional String field as illustrated below:

```
User Authentication
Condition String
(proxy.pathsuffix MatchesPath */SalesOrder*) AND (request.verb = "POST" OR request.verb = "PUT")
```

**Creating and Configuring Policies**

Adding a policy to an API proxy involves the following two steps:

1. Select an existing flow or create a conditional Flow
2. Create and attach the policy to the Flow
The above graphic illustrates the relationship between policies and Flows. A policy is attached to a Flow as a processing **Step**. Each Step can contain one policy. A flow can contain zero or many steps. Each step has a condition, which decides whether the policy has to be executed.

## 1.5.1.3.3 Condition Strings

Conditions enable API proxies to behave dynamically at runtime.

Conditions define operations on variables. Conditional statements are boolean and always evaluate to true or false. Developers commonly use both built-in flow variables and custom variables in conditional statements. The basic structure of a conditional statement is: `<Condition>{variable.name}{operator}{"value"}</Condition>`. In API Management, every API resource is treated as conditional flow. So for every resource, you can already see a condition defined in the policy designer.

Conditions can be chained. For example, the following condition evaluates to true only if the URI of the request matches `/statuses/user_timeline.json` and the HTTP verb of the request is GET.

**Example**

```
<Condition>(proxy.pathsuffix MatchesPath "/statuses/**") and (request.verb = "GET")</Condition>
```

**Usage of Conditions**

You can use conditions to control the following behavior in API Management:

1. Policy execution
2. Flow execution
3. Target endpoint route rule execution
Policy Execution

Using conditional statements, you can control the enforcement of policies. A common use case is conditional transformation of request/response messages, based on HTTP header or message content. For example, if you want to execute a key value map policy whenever the request has a query parameter called "country", you will attach the key value map policy to the required flow. To apply condition on this policy, you can add the following condition string in the Policy editor in the Condition string field: `request.queryparam.country IsNot null`.

Flow execution

Using conditional statements, you can control the execution of named flows in ProxyEndpoints and TargetEndpoints. Note that only 'named' flows can be executed conditionally. Preflows and postflows (both request and response) on ProxyEndpoints and TargetEndpoints execute for every transaction, and thus provide unconditional ‘failsafe’ capabilities.

Target endpoint route rule execution

Using conditional statements, you can control the target endpoint launched by proxy endpoint configuration. A route rule forwards a request to a particular target endpoint. When more than one target endpoint is available, the route rule is evaluated for its condition. If it is true, the request is forwarded to the named target endpoint. For example, if you want to restrict the service access to specific country, then you can add a route rule which has NONE as the target endpoint and the following condition string: `request.queryparam.country = "IN" Or request.queryparam.country = "DE"`.

For more information on how to define multiple target endpoints using Route Rule, see Enable Dynamic Routing [page 360].

Path Expressions

Path expressions are used for matching URI paths, using "*" to represent a single path element and "**" to represent multiple URI levels. For example:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Sample URI paths matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>/*/a/</td>
<td>/x/a/ or /y/a/</td>
</tr>
<tr>
<td>/<em>/a/</em></td>
<td>/x/a/b or /y/a/foo</td>
</tr>
<tr>
<td>/*/a/**</td>
<td>/x/a/b/c/d</td>
</tr>
<tr>
<td>/<em>/a/</em>/feed/</td>
<td>/x/a/b/feed/ or /y/a/foo/feed/</td>
</tr>
<tr>
<td>/a/<strong>/feed/</strong></td>
<td>/a/b/feed/rss/1234</td>
</tr>
</tbody>
</table>

% is treated as an escape character. The pattern %{user%} matches {user} but not user. Conditions can be categorized as follows:

1. **Operators**: Wildcard patterns used in conditions.
2. **Relational operands**: Evaluates whether a quantity is equal to, greater than, or less than another.
3. **Operands**: The value of the conditions is used to determine the overall values.

4. **Literals**: Literal values in a condition.

### Operators

When using operators, observe the following restrictions:

- Operators cannot be used as variable names.
- A space character is required before and after an operator.
- To include an operator in a variable, a variable name must be enclosed in single quotes. For example, "request.header.help!me".
- Arithmetic operators (+, *, -, /, %) are not supported.
- Java precedence is used for operators.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>In words (case insensitive)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>!</code></td>
<td>Not, not</td>
<td>Unary operator (takes a single input)</td>
</tr>
<tr>
<td><code>=</code></td>
<td>Equals, Is</td>
<td>Equals to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>NotEquals, IsNot</td>
<td>Not equals</td>
</tr>
<tr>
<td><code>:=</code></td>
<td>EqualsCaseInsensitive</td>
<td>Equals but is case insensitive</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>GreaterThan</td>
<td>Greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>GreaterThanOrEquals</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>LesserThan</td>
<td>Lesser than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>LesserThanOrEquals</td>
<td>Lesser than or equal to</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>And, and</td>
<td>And</td>
</tr>
<tr>
<td>`</td>
<td></td>
<td>`</td>
</tr>
<tr>
<td><code>(</code></td>
<td></td>
<td>Groups an expression</td>
</tr>
<tr>
<td><code>)</code></td>
<td></td>
<td>Closes an expression group</td>
</tr>
<tr>
<td><code>~~</code></td>
<td>JavaRegex</td>
<td>Matches a javax.util.regex compliant regular expression.</td>
</tr>
<tr>
<td><code>~</code></td>
<td>Matches, Like</td>
<td>Matches a glob-style pattern using the &quot;*&quot; wildcard character.</td>
</tr>
<tr>
<td><code>~/</code></td>
<td>MatchesPath, LikePath</td>
<td>Matches a path expression.</td>
</tr>
<tr>
<td>`=</td>
<td>`</td>
<td>StartsWith</td>
</tr>
</tbody>
</table>
Behavior of null operands in conditional statements

The following table shows the behavior when operands evaluate to null:

<table>
<thead>
<tr>
<th>Operator</th>
<th>LHS null</th>
<th>RHS null</th>
<th>LHS and RHS null</th>
</tr>
</thead>
<tbody>
<tr>
<td>=, ==, :=</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>!=</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>&lt;</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>&lt;=</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>~</td>
<td>false</td>
<td>N/A</td>
<td>false</td>
</tr>
<tr>
<td>~~</td>
<td>false</td>
<td>N/A</td>
<td>false</td>
</tr>
<tr>
<td>!~</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>~/</td>
<td>false</td>
<td>N/A</td>
<td>false</td>
</tr>
</tbody>
</table>

Operands

API Management adapts operands to a common data type before comparing them. For example, if the response status code is 404, the expression response.status.code = "400" and the response.status.code = 400 are equivalent. For numeric operands, the data type is interpreted as integer unless the value is terminated as follows:

- f" or "F" (float, for example, 3.142f, 91.1F)
- "d" or "D" (double, for example, 3.142d, 100.123D)
- "l" or "L" (long, for example, 12321421312L)

In these cases, the system performs the adaptations shown in the following table.

i Note
The dash character ( - ) in this table implies that the comparison is not performed, so the comparison is false.
null, true, and false are the literals available in conditions. For example: request.header.host is null and flow.cachehit is true.

1.5.1.3.4 Policies

Policy definition and types of policies supported by API Management.

API Management provides capabilities to define the behavior of an API by using 'policies.' A policy is a program that executes a specific function at runtime. They provide the flexibility to add common functionalities on an API without having to code them individually each time. Policies provide features to secure APIs, control the API traffic, and transform message formats. You can also customize the behavior of an API by adding scripts and attaching them to policies.

You can apply a policy on the request or response stream. You can also specify if it’s applicable on the proxy endpoint or target endpoint. For information on the types of policies supported by API Management, see Policy Types [page 93].

Defining Policies using Policy Designer

Use the policy designer to create policies. The set of prebuilt policies supported by API Management is available in the top-right pane. To create a policy, first select the Flow [page 85] segment on which this policy is
applicable. Then create the policy by adding the policy details in the editor. You also add a conditional string that ensures that the policy is executed only if the condition is met.

A sequence of policies can be applied on the desired Flow segment. The system executes the policies in the same order in which they’re applied. The list of policies created in the API Management is available in the bottom-right pane of the policy designer.

When you create a policy using the designer, you provide a name to the policy. Furthermore, mention whether it must be attached to the incoming or outgoing stream of the selected Flow.

There are few policies that work as expected only when associated with multiple flows.

For example, Response Cache policy must be attached to both request and response Flow of an API proxy. In such cases, you can add Response Cache policy to a request flow & then attach the same to the response flow.

To attach a policy to multiple flows, click "+" against the required policy in the Created Policies area.

### 1.5.1.3.4.1 Policy Types

Policies define a set of rules (such as, enforce security, control traffic, and so on) that is applied on the API.

Before you start defining policies, it is important to understand some common attributes that all policies share:

- **enabled**: This attribute determines whether a policy is switched on or off. Set this attribute to `true` to switch the policy on. A policy that has enabled set to `false` is not executed at runtime.
- **continueOnError**: Determines whether a policy should continue processing the message if the policy execution fails. For quota policies where the errors indicate that the policy limit has exceeded, this field should be set to `false`.
- **async**: Set `async=true` if you want the policy to run in a different thread that is isolated from the regular thread that services the request or response Flow.

The following is the list of prebuilt policies supported by API Management:

- Access Control
- Access Entity
- Assign Message
- Basic Authentication
- Concurrent Rate Limit
- Extract variables
- Invalidate Cache
- JavaScript
- JSON to XML
- Key Value Map Operations
- Lookup Cache
- Message Logging Policy
- OAuth v2.0
- OAuth v2.0 GET
- OAuth v2.0 SET
- Populate Cache
Restrict access to your APIs based on specific Identity Provisioning (IP) addresses.

This policy is used to selectively allow or deny access for an IP address or group of IP addresses. Use this policy when you want to limit access to APIs to only a specific IP address or group of IP addresses. For example, if you only want computers under the control of your enterprise to access the APIs exposed in your test environment, you can allow (allowlist) the IP address range for your internal network. Developers working from home can access these APIs using VPN.

You configure an Access Control policy as follows:

- Specify match rules for the two permitted actions (ALLOW or DENY).
  Specify the IP address (SourceAddress element) for each match rule. Also, configure a mask for each IP address. You allow or deny access based on a mask value on the IP address.

- Mention the order in which the rules are to be executed.
  The system executes the first matching rule in the defined order, and then subsequent matching rules are skipped. If the same rule is configured with both ALLOW and DENY actions, the rule that is defined first is executed and the subsequent rule is skipped.

You can attach this policy in the following locations:
How the Access Control policy determines which IP address to validate?

In an ideal scenario, IP addresses that are served can come from various sources in a request. For instance, the True-Client-IP header can contain an IP address and the X-Forwarded-For header can contain one or more IP addresses. Also, the API Management configuration and the policy configuration determine which X-Forwarded-For address(es) the policy evaluates.

This section describes how you can configure the Access Control policy to determine which IP address it chooses to validate. Following is the logic based on which Access Control policy determines the IP address it chooses to validate:

- **True-Client-IP header**
  The policy first checks if an IP address is present in the True-Client-IP header. If a valid IP address is present, the policy validates that IP address.

  ▲ Caution
  If you’re going to use the True-Client-IP header, then make sure that you trust the source of that address. If you can’t ensure that the header contains a trusted address, set <IgnoreTrueClientIPHeader> to true so that the policy ignores the True-Client-IP and instead evaluates the IP address(es) in the X-Forwarded-For header.

- **IgnoreTrueClientIPHeader**
  When you set <IgnoreTrueClientIPHeader> to true, the policy ignores the True-Client-IP header and evaluates IP addresses in the X-Forwarded-For header, following the behavior you’ve configured. When the IgnoreTrueClientIPHeader attribute is set to false, the policy evaluates the True-Client-IP header. By default, IgnoreTrueClientIPHeader attribute is set to false.

- **X-Forwarded-For header**
  If the True-Client-IP header doesn’t contain an IP address, or if you’ve set the IgnoreTrueClientIPHeader element to true, then the policy validates the IP addresses present in the -X-Forwarded-For header. If there are multiple addresses in the X-Forwarded-For header, then those IP addresses, likely belong to the chain of servers that processed a request.

  ▲ Note
  API Management, by default, fills the X-Forwarded-For header with a single IP address it received from the last external TCP handshake (such as the Client IP or router). That is, in API Management, the X-Forwarded-For header is populated with only a single IP address.

An example payload for the policy is as follows:

```xml
<!-- Use case-1 : Allow only a single IP -->
<AccessControl async='true' continueOnError='false' enabled='true'
xmllns='http://www.sap.com/apimgmt'>
  <IPRules noRuleMatchAction='DENY'>
    <MatchRule action='ALLOW'>
      <SourceAddress mask='32'>120.75.68.75</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>
<!-- Use case -2: Block a range of IP -->
<AccessControl async='true' continueOnError='false' enabled='true'
xmllns='http://www.sap.com/apimgmt'>
  <IPRules noRuleMatchAction='ALLOW'>
    <SourceAddress mask='32'>120.75.68.75</SourceAddress>
    <SourceAddress mask='32'>120.75.68.76</SourceAddress>
  </IPRules>
</AccessControl>
```
<MatchRule action='DENY'>
  <SourceAddress mask='8'>120.75.68.75</SourceAddress>
</MatchRule>
</IPRules>
</AccessControl>

<!-- Use case-3 : Allow a single IP from an identified range -->
<!-- In the below setting IP 120.75.68.75 is allowed and any other IP in the
range 120.75.68.* is blocked -->
<AccessControl async='true' continueOnError='false' enabled='true'
xmlns='http://www.sap.com/apimgmt'>
  <IPRules noRuleMatchAction='ALLOW'>
    <MatchRule action='ALLOW'>
      <SourceAddress mask='32'>120.75.68.75</SourceAddress>
    </MatchRule>
    <MatchRule action='DENY'>
      <SourceAddress mask='24'>120.75.68.75</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>

<!-- Use case-4 : The access control policy allows value from flow variables
-->  
<AccessControl async='true' continueOnError='false' enabled='true'
xmlns='http://www.sap.com/apimgmt'>
  <IPRules noRuleMatchAction='DENY'>
    <MatchRule action='ALLOW'>
      <SourceAddress mask='{{kvm.mask.value}}'>{{kvm.ip.value}}</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>

<!--kvm.mask.value and kvm.ip.value are read using Key Value Map Operations
policy before it's used in the
AccessControl policy-->

**Element and Attribute Descriptions**

<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPRules (Optional)</td>
<td>This is the parent element containing the rules to allow or deny IP addresses.</td>
</tr>
<tr>
<td>noRuleMatchAction (Mandatory)</td>
<td>Defines the action that has to be taken if the match rule isn’t resolved. This element contains child elements that define which IP addresses are permitted or restricted.</td>
</tr>
<tr>
<td></td>
<td>Valid value: ALLOW or DENY</td>
</tr>
<tr>
<td></td>
<td>The default value is ALLOW.</td>
</tr>
<tr>
<td></td>
<td>Syntax: <code>&lt;IPRules noRuleMatchAction = &quot;ALLOW&quot;&gt;</code></td>
</tr>
</tbody>
</table>
### Elements & Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchRule action (Mandatory)</td>
<td>Defines the action that has to be taken if the IP address matches the Source address defined. Valid value: ALLOW or DENY. The default value is ALLOW.</td>
</tr>
<tr>
<td>SourceAddress (Optional)</td>
<td>This element indicates the IP address range of a client. The valid IP address of the consumer in dotted decimal notation is a valid value. For example, 127.0.0.1.</td>
</tr>
<tr>
<td>Mask (Mandatory)</td>
<td>Use this attribute in conjunction with the SourceAddress element. The mask refers to the number of bits in the IP Address that has to be considered. The maximum value of mask is less than or equal to 32. For example:</td>
</tr>
</tbody>
</table>

#### Sample Code

**Example**

```xml
<IPRules noRuleMatchAction = "ALLOW">
  <MatchRule action = "ALLOW">
    <SourceAddress mask="32">120.75.68.75</SourceAddress>
  </MatchRule>
  <MatchRule action = "DENY">
    <SourceAddress mask="24">120.75.68.75</SourceAddress>
  </MatchRule>
</IPRules>
```

Then, all the IP Addresses with the pattern `20.10.*` are allowed to access the proxy.

### During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Cause</th>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ClientIpExtractionFailed</td>
<td>See fault string.</td>
</tr>
</tbody>
</table>
## Error Name

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPDeniedAccess</td>
<td>See fault string.</td>
</tr>
<tr>
<td>InvalidIPAddress</td>
<td>See fault string.</td>
</tr>
<tr>
<td>InvalidIPv4Address</td>
<td>See fault string.</td>
</tr>
<tr>
<td>InvalidIPv6Address</td>
<td>See fault string.</td>
</tr>
<tr>
<td>InvalidRulePattern</td>
<td>See fault string.</td>
</tr>
</tbody>
</table>

Following fault variables are set when the policy triggers an error at runtime:

### Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The [prefix] is acl. The [policy_name] is the name of the policy that threw the error.</td>
<td>acl.AC-AllowAccess.failed = true</td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name = &quot;IPDeniedAccess&quot;</td>
</tr>
</tbody>
</table>

Following is an example of an error response:

```json
{
  "fault":{
    "detail":{
      "errorcode":"steps.accesscontrol.IPDeniedAccess"
    },
    "faultstring":"Access Denied for client ip : 51.218.253.1"
  }
}
```

Following is an example of a fault rule:

```xml
<FaultRule name="IPDeniedAccess">
  <Step>
    <Name>AssignMsg-IPDeniedAccess</Name>
    <Condition>(fault.name Matches "IPDeniedAccess")</Condition>
  </Step>
  <Condition>(acl.failed = true)</Condition>
</FaultRule>
```
1.5.1.3.4.1.2 Access Entity

API Management stores profile data for a range for "entities", such as developers, applications, and API products. The Access Entity policy enables developers to retrieve those profiles during API proxy message processing. As such, the Access Entity policy functions as a policy-based runtime database lookup. The profile information returned by this policy can be used to enable dynamic behavior, such as conditional endpoint routing, flow execution, policy enforcement, and so on.

For example, you could use the Access Entity policy to get the profile for an app, and then extract a custom field (such as a department name) from the app profile. Using the department name as a variable, you could route the request to a specific backend service, or you could forward the department name to Analytics to enable data accounting.

When a policy of type Access Entity is enforced:

1. The policy sets an entity as an XML-formatted flow variable. The variable that is set is usually consumed by an Extract Variable or Assign Message policy.
2. XPath is used to parse the desired properties from the profile.
3. If the specified entity is not found, the policy returns ResourceNotFoundException.

Access Entity can be used to access profiles for the following entities:

- Application
- API product
- Consumer key
- Developer
- Company
- Company developer

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request →</td>
<td></td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
<tr>
<td></td>
<td>←Response</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<!-- Use case 1 : Access developer from the current apikey which arrives in the request header -->
<AccessEntity async="true" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <EntityType value="developer"/>
  <EntityIdentifier ref="request.header.apikey" type="consumerkey"/>
</AccessEntity>
```
For the above use case, if the policy is named as "AccessDeveloper" then a flow variable named "AccessEntity.AccessDeveloper" will hold the details of the developer in xml format. An extract variable policy can be used to extract any field from the developer details. Mentioned Below is an example to extract the developer e-mail into a flow variable named "developerEmail".

**Sample Code**

```xml
<ExtractVariables async="true" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Source>AccessEntity.AccessDeveloper</Source>
  <XMLPayload>
    <Variable name="developerEmail" type="string">
      <!-- Specifies the XPath defined for the variable -->
      <XPath>/Developer/Email</XPath>
    </Variable>
  </XMLPayload>
</ExtractVariables>
```

**Sample Code**

```xml
<!–- Use case 2 : Access product details from the current apikey which arrives in the request header
If the value for EntityType is changed to apiproduct, associated API product will be fetched populated in AccessEntity.{policy_name} flow variable. -->
<AccessEntity async="true" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <EntityType value="apiproduct"/>
  <EntityIdentifier ref="request.header.apikey" type="consumerkey"/>
</AccessEntity>
```

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityType (Mandatory)</td>
<td>The element indicates the type of entity to be retrieved from the data store. The permitted values for this element are provided in the table below. Syntax: &lt;EntityType value=&quot;entity_type&quot;/&gt;</td>
</tr>
</tbody>
</table>
## Elements and Attributes

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EntityIdentifier (Mandatory)</strong></td>
</tr>
<tr>
<td>The value that identifies the specific entity whose profile should be retrieved.</td>
</tr>
<tr>
<td>The ref attribute identifies the variable that provides the source of the identifier, for example, <code>request.queryparam.apikey</code>.</td>
</tr>
<tr>
<td>The type attribute identifies the EntityType populated by the referenced variable, such as <code>consumerkey</code>.</td>
</tr>
</tbody>
</table>

**Syntax:**
```
<EntityIdentifier ref="value_variable" type="identifier_type" />
```

### Sample Code

**Example**
```
<?xml version="1.1" encoding="UTF-1" standalone="yes"?>
<AccessEntity async="true" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <DisplayName>FetchCompanyProfile</DisplayName>
  <EntityType value="company"></EntityType>
  <EntityIdentifier ref="request.queryparam.apikey" type="appid"/>
</AccessEntity>
```
Elements and Attributes

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecondaryIdentifier (Optional)</td>
</tr>
<tr>
<td>This element is optional but if used, ref and type are mandatory.</td>
</tr>
<tr>
<td>Use this element when the EntityIdentifier does not return a unique value,</td>
</tr>
<tr>
<td>for example, appname. You cannot use multiple SecondaryIdentifier elements.</td>
</tr>
<tr>
<td>The ref attribute identifies the variable that provides the source of the</td>
</tr>
<tr>
<td>identifier, for example, request.query-param.apikey.</td>
</tr>
<tr>
<td>The type identifies the entity type populated by the referenced variable,</td>
</tr>
<tr>
<td>such as consumerkey. The use of multiple SecondaryIdentifier elements is</td>
</tr>
<tr>
<td>not supported.</td>
</tr>
<tr>
<td>Syntax: &lt;SecondaryIdentifier ref=&quot;value_variable&quot; type=&quot;identifier_type&quot;&gt;</td>
</tr>
</tbody>
</table>

Sample Code

Example

```xml
<?xml version="1.1" encoding="UTF-1" standalone="yes"?
&gt;&lt;AccessEntity async="true" continueOnError="false"
enabled="true" xmlns='http://www.sap.com/apimgmt'
 &lt;DisplayName>FetchCompanyProfile</DisplayName>
 &lt;EntityType value="company"></EntityType>
 &lt;EntityIdentifier ref="developer.app.name" type="appname"/
 &lt;SecondaryIdentifier ref="developer.id" type="developerid"/>
 &lt;/AccessEntity>
```

The following table illustrates the values supported for Entity Type elements:

<table>
<thead>
<tr>
<th>Entity Type Value</th>
<th>EntityIdentifier Types</th>
<th>SecondaryIdentifier Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiproduct</td>
<td>apid</td>
<td>apiresource</td>
</tr>
<tr>
<td></td>
<td>apiproductname</td>
<td></td>
</tr>
<tr>
<td>Entity Type Value</td>
<td>Entity Identifier Types</td>
<td>Secondary Identifier Types</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>appname</td>
<td></td>
<td>apiresource</td>
</tr>
<tr>
<td></td>
<td></td>
<td>developeremail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>developerid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>companyname</td>
</tr>
<tr>
<td>consumerkey</td>
<td></td>
<td>apiresource</td>
</tr>
<tr>
<td>app</td>
<td>appid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>appname</td>
<td>developeremail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>developerid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>companyname</td>
</tr>
<tr>
<td>authorizationcode</td>
<td>authorizationcode</td>
<td></td>
</tr>
<tr>
<td>company</td>
<td>appid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>company</td>
<td></td>
</tr>
<tr>
<td></td>
<td>consumerkey</td>
<td></td>
</tr>
<tr>
<td>companydeveloper</td>
<td>companyname</td>
<td></td>
</tr>
<tr>
<td>consumerkey</td>
<td>consumerkey</td>
<td></td>
</tr>
<tr>
<td>consumerkey_scope</td>
<td>consumerkey</td>
<td></td>
</tr>
<tr>
<td>developer</td>
<td>appid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>consumerkey</td>
<td></td>
</tr>
<tr>
<td></td>
<td>developeremail</td>
<td></td>
</tr>
<tr>
<td></td>
<td>developerid</td>
<td></td>
</tr>
<tr>
<td>requesttoken</td>
<td>requesttoken</td>
<td>consumerkey</td>
</tr>
<tr>
<td>verifier</td>
<td>verifier</td>
<td></td>
</tr>
</tbody>
</table>

**Related Information**

- [Assign Message](#)
- [Extract Variables](#)
1.5.1.3.4.1.3 Assign Message

This policy allows you to create new or modify an existing HTTP request or response message. You can create a new request message to a target service which will be invoked in a Service Callout policy within the API Proxy execution flow or modify the existing request or response messages to the backend service.

This policy is so named because you need to assign a message to a variable. To use the Assign Message policy, you must select a variable name and specify the message content to assign to it. If you choose to use the standards names such as request or response, the value will be assigned to the request or response flows. If you use any other name, it will refer to a custom variable that can exist within the API Proxy execution flow.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
</table>

An example payload for the policy is as follows:

```xml
<AssignMessage async="true" continueOnError="true" enabled="true">
  <Copy source="{source}">
    <Headers>
      <Header name="{header_name}"/>
    </Headers>
    <QueryParams>
      <QueryParam name="{query_param_name}"/>
    </QueryParams>
    <FormParams>
      <FormParam name="{form_param_name}"/>
    </FormParams>
    <Payload>{boolean_value}</Payload>
    <Verb>{boolean_value}</Verb>
    <Version>{boolean_value}</Version>
    <Path>{boolean_value}</Path>
    <StatusCode>{boolean_value}</StatusCode>
    <ReasonPhrase>{boolean_value}</ReasonPhrase>
  </Copy>
  <Remove>
    <Headers>
      <Header name="{header_name}"/>
    </Headers>
    <QueryParams>
      <QueryParam name="{query_param_name}"/>
    </QueryParams>
    <FormParams>
      <FormParam name="{form_param_name}"/>
    </FormParams>
    <Payload>{boolean_value}</Payload>
  </Remove>
  <Add>
    <Headers>
      <Header name="{header_name}">{value} />
    </Headers>
  </Add>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignTo (Optional)</td>
<td>Specifies the variable to which the message will be assigned. If this element is missing, it is treated as request or response, depending on the Flow to which the policy is attached. If attached to a response Flow, for example, the default type is response. In some cases, you cannot change the object on which this policy works. For example, you cannot change query parameters or form parameters on the response, but can only do so on the request.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <IgnoreUnresolvedVariables>[true|false]</IgnoreUnresolvedVariables>
  <AssignTo createNew="[true|false]" transport="http" type="[request|response]">destination_variable_name</AssignTo>
</AssignMessage>
```

#### Example

The following example specifies that the target is the original request that will be sent to the target endpoint:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request">destination_variable_name</AssignTo>
</AssignMessage>
```
### Elements & Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createNew (Optional)</td>
<td>It is a Boolean value which indicates if the request or response object should be newly created or if the existing message should be modified. If the value is <code>true</code>, the policy creates a new request or response object, based on the type specified. If no name is specified for the new variable, the policy creates a new request or response object, based on the value of <code>type</code>.</td>
</tr>
<tr>
<td>transport (Optional)</td>
<td>It is a string which indicates the transport method for request and response messages. The only supported value is HTTP.</td>
</tr>
<tr>
<td>type (Optional)</td>
<td>It is a string that specifies the type of the new message, when <code>createNew</code> is <code>true</code>. Valid values: <code>request</code> or <code>response</code>. Default value: <code>request</code>.</td>
</tr>
<tr>
<td>Elements &amp; Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IgnoreUnresolvedVariables (Optional)</td>
<td>If <strong>IgnoreUnresolvedVariables</strong> is set to <strong>false</strong> and any variable cannot be resolved, then the policy throws an error. If it is set to <strong>true</strong> and any variable is unresolvable, the variable is treated as empty string (Null). Valid values: true or false. Default value: false.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <IgnoreUnresolvedVariables>[true|false]</IgnoreUnresolvedVariables>
</AssignMessage>
```

#### Example

The following example sets **IgnoreUnresolvedVariables** to **true**:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="response">
    ...
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  </Copy>
</AssignMessage>
```

<table>
<thead>
<tr>
<th>Copy (Optional)</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Copies the specified information from the <code>&lt;source&gt;</code> to the variable specified in the <code>&lt;AssignTo&gt;</code> element. If the source is not specified, it is treated as a simple message. If the source variable cannot be resolved, or resolves to a non-message type, <code>&lt;Copy&gt;</code> fails to respond.</td>
</tr>
</tbody>
</table>
### Elements & Attributes

<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Headers</td>
<td>Copies HTTP headers. To copy multiple headers, mention the header name in the name attribute as below: <code>&lt;Copy source='request'&gt; &lt;Headers&gt; &lt;Header name=&quot;headerA&quot;/&gt; &lt;Header name=&quot;headerB&quot;/&gt; &lt;/Headers&gt; &lt;/Copy&gt;</code> To copy all headers, specify <code>&lt;Copy&gt;&lt;Headers/&gt;&lt;/Copy&gt;</code>.</td>
</tr>
</tbody>
</table>

#### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'> 
  <Copy source="[request|response]">  
    <!-- Can also be an empty array (<Headers/>) --> 
    <Headers>  
      <Header name="header_name">header_value</Header> ...  
    </Headers>  
  </Copy>  
  <IgnoreUnresolvedVariables>[true|false]</IgnoreUnresolvedVariables> </AssignMessage>
```

**Example**

The following example copies the temp header from the request to the new CustomReq object:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'> 
  <Copy source="request">  
    <Headers>  
      <Header name="temp"/>  
    </Headers>  
  </Copy>  
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables> 
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;AssignTo createNew=&quot;true&quot; transport=&quot;http&quot; type=&quot;request&quot;&gt;CustomReq&lt;/AssignTo&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;/AssignMessage&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>
Elements & Attributes

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryParams</td>
</tr>
</tbody>
</table>

Copies query parameters.

Note that the QueryParams is copied only when both the source and AssignTo type are request.

To copy all query parameters, specify `<Copy><QueryParams/></Copy>`.

You can use query parameters only when the message type is Request and the HTTP verb is GET.

**Sample Code**

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="[request|response]">
    <!-- Can also be an empty array (<QueryParams/>) -->
    <QueryParams>
      <QueryParam name="queryparam_name">queryparam_value</QueryParam>
      ...
    </QueryParams>
  </Copy>
  <IgnoreUnresolvedVariables>[true|false]</IgnoreUnresolvedVariables>
</AssignMessage>
```

**Example**

The following example copies the `temp_param` query parameter from the request into a new CustomReq object:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="request">
    <QueryParams>
      <QueryParam name="temp_param"/>
    </QueryParams>
  </Copy>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="request">CustomReq</AssignTo>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
</table>
| FormParams           | Copies the form parameters from the request specified in the `<source>` attribute of `<Copy>` to the request specified by `AssignTo`. Note that the FormParams is copied only when the contentType is `source` and `AssignTo` is `application/x-wwwform-urlencoded`. To copy all form parameters, specify `<Copy><FormParams/></Copy>`. You can use query parameters only when the message type is `Request` and the HTTP verb is `POST`. Additionally, you should meet one (or both) of the following criteria:  
  - Set the Form data to some value, or `' ' ` (empty string). For example, with `curl`, add `-d ''` to your request.  
  - Set the `Content-Length` header to `0` if there is no data in the original request; otherwise, set it to the current length in bytes. For example, with `curl`, add `-H "Content-Length: 0"` to your request. |

**Sample Code**

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="[request|response]">
    <!-- Can also be an empty array (<FormParams/>) -->
    <FormParams>
      <FormParam name="formparam_name">formparam_value</FormParam>
      ...
    </FormParams>
    </Copy>
</AssignMessage>
```

**Example**

The following example copies three form parameters to the custom request `CustomReq`:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="request">
    <FormParams>
      <FormParam name="pName1"/>
      <FormParam name="pName2"/>
      <FormParam name="pName3"/>
    </FormParams>
  </Copy>
</AssignMessage>
```
### Elements & Attributes

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;FormParams&gt;</td>
</tr>
<tr>
<td>&lt;/Copy&gt;</td>
</tr>
<tr>
<td>&lt;IgnoreUnresolvedVariables&gt;false&lt;/IgnoreUnresolvedVariables&gt;</td>
</tr>
<tr>
<td>&lt;AssignTo createNew=&quot;true&quot; transport=&quot;http&quot; type=&quot;request&quot;&gt;CustomReq&lt;/AssignTo&gt;</td>
</tr>
<tr>
<td>&lt;/AssignMessage&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid values: true or false.</td>
</tr>
</tbody>
</table>

If true, the Content-Type header is copied.

#### Sample Code

**Syntax**

```xml
&lt;AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  &lt;Copy source="[request|response]"> |
  &lt;Payload>[false|true]</Payload&gt; |
  &lt;/Copy&gt; |
&lt;/AssignMessage&gt;
```

**Example**

The following example sets &lt;Payload&gt; to “true” so that the request payload is copied from the request to the response:

```xml
&lt;AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  &lt;Copy source="request"> |
  &lt;Payload>true</Payload&gt; |
  &lt;/Copy&gt; |
&lt;/AssignMessage&gt;
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td>Valid values: true or false. If true, the HTTP version is copied.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="[request|response]">
    <Version>[false|true]</Version>
  </Copy>
</AssignMessage>
```

#### Example

The following example sets `<Version>` to "true" on the request, which copies the version from the default request object to a new custom request object:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="request">
    <Version>true</Version>
  </Copy>
  <AssignTo createNew="true" transport="http" type="request">CustomReq</AssignTo>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Verb</strong></td>
<td>Valid values: true or false. If true, the verb of the request gets assigned to the new request message, which is indicated by the AssignTo variable. This element is applicable only for HTTP request and not for response.</td>
</tr>
</tbody>
</table>

### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="[request|response]">
    <Verb>[false|true]</Verb>
  </Copy>
</AssignMessage>
```

**Example**

The following example sets `<Verb>` to "true", which copies the verb from the default request to a new custom request:

```xml
<AssignMessage name="copy-verb-1">
  <Copy source="request">
    <Verb>true</Verb>
  </Copy>
  <AssignTo createNew="true" transport="http" type="request">MyCustomRequest</AssignTo>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>If true, the path of the request gets assigned to the path of the new request object, which is indicated by the AssignTo variable. This element is applicable only for HTTP request and not for response.</td>
</tr>
</tbody>
</table>

**Sample Code**

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="[request|response]">
    <Path>true</Path>
  </Copy>
</AssignMessage>
```

**Example**

The following example indicates that Assign Message should copy the path from the source request to the new custom request object:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="request">
    <Path>true</Path>
  </Copy>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="request">CustomReq</AssignTo>
</AssignMessage>
```
### Elements & Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StatusCode</td>
<td>Valid values: true or false. If true, the response status gets assigned to the new response message, which is indicated by the AssignTo variable. This element is applicable only for HTTP request and not for response.</td>
</tr>
</tbody>
</table>

#### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="[request|response]">
    <StatusCode>[false|true]</StatusCode>
  </Copy>
</AssignMessage>
```

**Example**

The following example sets `<StatusCode>` to "true", which copies the status code from the default response object to a new custom response object:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="response">
    <StatusCode>true</StatusCode>
  </Copy>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="response">CustomReq</AssignTo>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason Phrase</td>
<td>If true, the reason phrase of the response gets assigned to the new response message, which is indicated by the AssignTo variable. This element is applicable only for HTTP request and not for response.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="[request|response]">
    <ReasonPhrase>true</ReasonPhrase>
  </Copy>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="response">CustomReq</AssignTo>
</AssignMessage>
```

#### Example

The following example sets `<ReasonPhrase>` to "true", which causes `<Copy>` to copy the reason phrase from the default response to a custom response object:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Copy source="response">
    <ReasonPhrase>true</ReasonPhrase>
  </Copy>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="response">CustomReq</AssignTo>
</AssignMessage>
```
### Elements & Attributes

<table>
<thead>
<tr>
<th>Remove (Optional)</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td>Removes HTTP headers from the variable specified in the <code>&lt;AssignTo&gt;</code> element. To remove all the headers, specify <code>&lt;Remove&gt;&lt;Headers/&gt;&lt;/Remove&gt;</code>. To remove specific headers, provide the header name in the name attribute as below: <code>&lt;Remove&gt;&lt;Headers&gt;&lt;Header name=&quot;headerA&quot;/&gt;&lt;/Header&gt;&lt;Header name=&quot;headerB&quot;/&gt;&lt;/Headers&gt;&lt;/Remove&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

---

#### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Remove>
    <!-- Can also be an empty array (/<Headers/>) -->
    <Headers>
      <Header name="header_name">header_value</Header>
      ...
    </Headers>
  </Remove>
</AssignMessage>
```

**Example**

The following example removes the `temp` header from the request:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Remove>
    <Headers>
      <Header name="temp"/>
    </Headers>
  </Remove>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request"/>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
</table>
| QueryParams           | Removes the query parameters. Note that the QueryParams are removed only when the AssignTo type is request and the HTTP verb is GET. To remove all query parameters, specify `<Remove><QueryParams/></Remove>`.

### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Remove>  <!-- Can also be an empty array (QueryParams/) -->  
    <QueryParams>  
      <QueryParam name="queryparam_name">queryparam_value</QueryParam>  
    ...  
    </QueryParams>  
  </Remove>  
</AssignMessage>
```

**Example**

The following example removes all query parameters from the request:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Remove>  
    <QueryParams/>  
  </Remove>  
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>  
  <AssignTo createNew="false" transport="http" type="request"/></AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormParams</td>
<td>Removes the form parameters.</td>
</tr>
</tbody>
</table>

Note that the FormParams are removed only when the contentType of AssignTo is application/x-www-form-urlencoded. To remove all query parameters, specify `<Remove><FormParams/></Remove>`.

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Remove>
    <!-- Can also be an empty array (<FormParams/>) -->
    <FormParams>
      <FormParam name="form_param_name">formparam_value</FormParam>
      ...
    </FormParams>
  </Remove>
</AssignMessage>
```

#### Example

The following example removes three form parameters from the request:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Remove>
    <FormParams>
      <FormParam name="form_param_1"/>
      <FormParam name="form_param_2"/>
      <FormParam name="form_param_3"/>
    </FormParams>
  </Remove>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="application/x-www-form-urlencoded"/>
</AssignMessage>
```
### Elements & Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload</td>
<td>Valid values: <code>true</code> or <code>false</code>. If true, the payload is cleared.</td>
</tr>
</tbody>
</table>

#### Syntax

```xml
<AssignMessage async="false|true"
    continueOnError="[true|false]"
    enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
    <Remove>
        <Payload>[false|true]</Payload>
    </Remove>
</AssignMessage>
```

#### Example

The following example sets `<Payload>` to “true” so that the request payload is cleared:

```xml
<AssignMessage async="false"
    continueOnError="false" enabled="true"
    xmlns='http://www.sap.com/apimgmt'>
    <Remove>
        <Payload>true</Payload>
    </Remove>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
    <AssignTo createNew="false"
        transport="http" type="request"/>
</AssignMessage>
```
Add (Optional) Headers

Adds the headers in the variable specified in the <AssignTo> element.

Note that the empty header <Add><Headers/></Add> does not add any header. The same holds true for QueryParams and FormParams.

**Sample Code**

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Add>
    <Headers>
      <Header name="header_name">header_value</Header>
    </Headers>
  </Add>
</AssignMessage>
```

**Example**

The following example adds the temp header to the request message, and assigns the value of the request.temp flow variable to that header:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Add>
    <Headers>
      <Header name="temp">{request.temp}</Header>
    </Headers>
  </Add>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryParams</td>
<td>Adds the query parameters. You can use query parameters only when the message type is Request and the HTTP verb is GET.</td>
</tr>
</tbody>
</table>

### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Add>
    <QueryParams>
      <QueryParam name="queryparam_name">queryparam_value</QueryParam>
      ...
    </QueryParams>
  </Add>
</AssignMessage>
```

**Example**

The following example adds the query parameter `tempParam` to the request and assigns the value `82` to it:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Add>
    <QueryParams>
      <QueryParam name="tempParam">82</QueryParam>
    </QueryParams>
  </Add>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
</table>
| FormParams           | Adds the form parameters and the contentType of message is changed to application/x-www-form-urlencoded. You can use form parameters only when the message type is Request and the HTTP verb is POST. Additionally, you should meet one (or both) of the following criteria:  
  - Set the Form data to some value, or ' ' (empty string). For example, with `curl`, add `-d ''` to your request.  
  - Set the `Content-Length` header to 0 if there is no data in the original request; otherwise, set it to the current length in bytes. For example, with `curl`, add `-H "Content-Length: 0"` to your request. |

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Add>
    <FormParams>
      <FormParam name="formparam_name">formparam_value</FormParam>
      ...
    </FormParams>
    <AssignTo createNew="[true|false]" transport="http" type="[request|response]">destination_variable_name</AssignTo>
  </Add>
</AssignMessage>
```

#### Example

The following example adds a single form parameter ("answer") and a static value ("42") to the request:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Add>
    <FormParams>
      <FormParam name="answer">42</FormParam>
    </FormParams>
  </Add>
</AssignMessage>
```
### Elements & Attributes

<table>
<thead>
<tr>
<th>Set (Optional)</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets or overwrites the HTTP headers in the variable specified in the &lt;AssignTo&gt; element. Note that the empty header &lt;Set&gt;&lt;Headers/&gt;&lt;/Set&gt; does not set any header. The same holds true for QueryParams and FormParams.</td>
</tr>
</tbody>
</table>

#### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <Headers>
      <Header name="header_name">header_value</Header>
      ...
    </Headers>
  </Set>
</AssignMessage>
```

**Example**

The following example sets the user-agent header to the value of the request.header.user-agent variable:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Headers>
    <Header name="user-agent">{request.header.user-agent}</Header>
  </Headers>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="response"/>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryParams</td>
<td>Sets or overwrites the query parameters in a request.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <QueryParams>
      <QueryParam name="queryparam_name">queryparam_value</QueryParam>
      ...
    </QueryParams>
  </Set>
</AssignMessage>
```

#### Example

The following example sets the "temp" query parameter to the value of the request.header.temp variable:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <QueryParams>
    <QueryParam name="temp">{request.header.temp}</QueryParam>
    ...
  </QueryParams>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormParams</td>
<td>Sets or overwrites the form parameters and the content type of message changes to application/x-www-form-urlencoded. You can use form parameters only when the message type is Request and the HTTP verb is POST.</td>
</tr>
</tbody>
</table>

**Sample Code**

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <FormParams>
      <FormParam name="formparam_name">formparam_value</FormParam>
      ...
    </FormParams>
  </Set>
</AssignMessage>
```

**Example**

The following example sets a form parameter called "tempparam" to the value of the request.header.tempparam variable in a new custom request:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <FormParams>
    <FormParam name="tempparam">{request.header.tempparam}</FormParam>
  </FormParams>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="request">Custom Req</AssignTo>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload</td>
<td>Enter the payloads of type json, xml, plain text, and so on, within this element.</td>
</tr>
</tbody>
</table>

Following are the attributes (optional) of `<Payload>`:

- **contentType**: It is a string which if specified, assigns the value of `contentType` to the Content-Type HTTP header.
- **variablePrefix**: It is a character which optionally specifies the leading delimiter on a flow variable. The default is `{`.
- **variableSuffix**: It is a character which optionally specifies the trailing delimiter on a flow variable. The default is `}`.

### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <Payload contentType="content_type" variablePrefix="prefix"
               variableSuffix="suffix">new_payload</Payload>
  </Set>
</AssignMessage>
```

**Example**

The following example sets a JSON payload:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <Payload contentType="application/json">
      {"name":"foo", "type":"bar"}
    </Payload>
  </Set>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Sets the HTTP version on a request.</td>
</tr>
</tbody>
</table>

**Sample Code**

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="false|true" enabled="false|true" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <Version>[1.0|1.1|{variable}]</Verb>
  </Set>
</AssignMessage>
```

**Example**

The following example uses a variable in curly braces to set the version number:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <Version>{my_version}</Verb>
  </Set>
</AssignMessage>
```

The content of `<Version>`, wrapped in curly braces is a message template and is replaced at runtime with the value of the referenced variable.
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason Phrase</td>
<td>Sets the reason phrase only when AssignTo type is response. This is generally used in combination with the status code for debugging.</td>
</tr>
</tbody>
</table>

### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <ReasonPhrase>reason_for_error or {variable}</ReasonPhrase>
  </Set>
</AssignMessage>
```

**Example**

The following example uses a variable to populate a reason phrase:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <ReasonPhrase>{calloutresponse.reason.phrase}</ReasonPhrase>
  </Set>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="response"/>
</AssignMessage>
```

The content of `<ReasonPhrase>`, wrapped in curly braces is a message template and is replaced at runtime with the value of the referenced variable.
## Elements & Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status Code</strong></td>
<td>Sets the status code only when AssignTo type is response.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>`&lt;AssignMessage async=&quot;false</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;Set&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;StatusCode&gt;HTTP_status_code or (variable)&lt;/StatusCode&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;/Set&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;/AssignMessage&gt;</code></td>
</tr>
<tr>
<td><strong>Verb</strong></td>
<td>Sets the HTTP verb and path only when AssignTo type is request.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>`&lt;AssignMessage async=&quot;false</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;Set&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>`&lt;Verb&gt;[GET</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;/Set&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;/AssignMessage&gt;</code></td>
</tr>
<tr>
<td>Elements &amp; Attributes</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td><code> xmlns='http://www.sap.com/apimgmt'</code></td>
<td></td>
</tr>
</tbody>
</table>

### Sample Code

**Example**

The following example uses a variable to populate a verb:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <Verb>{my_variable}</Verb>
  </Set>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http" type="request"/>
</AssignMessage>
```

The content of `<Verb>`, wrapped in curly braces is a message template and is replaced at runtime with the value of the referenced variable.
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignVariable</td>
<td>Name (Required) It is a string that specifies the name of the variable. If the variable named in AssignVariable does not exist, the policy creates one with that name.</td>
</tr>
</tbody>
</table>

### Sample Code

**Syntax**

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
    <AssignVariable>
        <Name>variable_name</Name>
    </AssignVariable>
</AssignMessage>
```

**Example**

The following example specifies the destination variable as `var` and sets it to the value "83":

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
    <AssignVariable>
        <Name>var</Name>
        <Value>83</Value>
    </AssignVariable>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
    <AssignTo createNew="true" transport="http" type="request"/>
</AssignMessage>
```

If `myvar` does not exist, AssignVariable creates it.
Elements & Attributes

<table>
<thead>
<tr>
<th>Description</th>
<th>Ref (Optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference that assigns value (as a flow variable and not a string variable) to the variable.</td>
<td></td>
</tr>
<tr>
<td>If you want to assign a literal string value to the variable, use the Value element instead.</td>
<td></td>
</tr>
<tr>
<td>Do this (no brackets): (&lt;\text{Ref}&gt;\text{client.host}&lt;/\text{Ref}&gt;)</td>
<td></td>
</tr>
<tr>
<td>Do NOT do this (brackets): (&lt;\text{Ref}&gt;{\text{client.host}}&lt;/\text{Ref}&gt;)</td>
<td></td>
</tr>
</tbody>
</table>

Define the default value for the destination flow variable by using \(<\text{Value}>\) along with \(<\text{Ref}>\). If the flow variable specified by \(<\text{Ref}>\) does not exist, is not readable, or is null, then the value of \(<\text{Value}>\) is assigned to the destination flow variable instead.

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
    <AssignVariable>
        <Name>variable_name</Name>
        <Ref>source_variable</Ref>
    </AssignVariable>
</AssignMessage>
```

#### Example

The following example assigns the value of the flow variable `request.header.temp` to the destination flow variable `var` and the value of the query parameter `test` to the `test` variable:

```xml
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
    <AssignVariable>
        <Name>var</Name>
        <Ref>request.header.temp</Ref>
    </AssignVariable>
    <AssignVariable>
        <Name>test</Name>
        <Ref>request.queryparam.test</Ref>
    </AssignVariable>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
    <AssignTo createNew="true" transport="http" type="request"/>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template (Optional)</td>
<td>It is a string that specifies the message template, which support functions like escaping and case conversion.</td>
</tr>
</tbody>
</table>

**Sample Code**

**Syntax**
```
<AssignMessage async="false|true" continueOnError="[true|false]" enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <AssignVariable>
    <Template>message_template</Template>
  </AssignVariable>
</AssignMessage>
```

**Example**

The following example uses the message template syntax to concatenate two context variables with a literal string (hyphen) between them:
```
<AssignMessage name='template-1'>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignVariable>
    <Name>my_destination_variable</Name>
    <Value>BADDBEF</Value>
    <Template>{system.uuid}-{messageid}</Template>
  </AssignVariable>
</AssignMessage>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value (Optional)</td>
<td>It is a string that specifies the value of the variable. If you use a combination of the <code>&lt;Value&gt;</code> and <code>&lt;Ref&gt;</code> elements, <code>&lt;Value&gt;</code> acts as default. If <code>&lt;Ref&gt;</code> is not specified or is unresolvable, this literal string value is assigned to the variable.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Syntax

```xml
<AssignMessage async="false|true" continueOnError="[true|false]
enabled="[true|false]" xmlns='http://www.sap.com/apimgmt'>
  <AssignVariable>
    <Name>variable_name</Name>
    <Value>variable_value</Value>
  </AssignVariable>
</AssignMessage>
```

#### Example

The following example assigns the value of the flow variable `request.header.temp` to the destination flow variable `var` and the value of the query parameter `test` to the `test` variable:

```xml
<AssignMessage name="assignvariable-2">
  <AssignVariable>
    <Name>var</Name>
    <Value>ErrorOnCopy</Value>
  </AssignVariable>
  <AssignVariable>
    <Name>test</Name>
    <Value>ErrorOnCopy</Value>
  </AssignVariable>
</AssignMessage>
```

If either assignment fails, the value "ErrorOnCopy" is assigned to the variable.

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnresolvedVariable</td>
<td>A flow variable referenced in the policy does not exist. Be sure that the variable is in scope - some of the built-in variables are only available in certain flow contexts.</td>
</tr>
<tr>
<td>Error Name</td>
<td>Cause</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VariableOfNonMsgType</td>
<td>The policy tried to assign a value to a non-message type variable. Message type variables include request and response. They also can be custom variables that are of type message. You might see this in the &lt;Copy&gt; element if you set the source attribute to a variable that is not of type Message.</td>
</tr>
<tr>
<td>SetVariableFailed</td>
<td>The policy was not able to set a variable. See the fault string for the name of the unresolved variable</td>
</tr>
<tr>
<td>InvalidIndex</td>
<td>The index must be greater than zero when specified in the Copy and Remove operations. For example, a query parameter can have multiple values. This error occurs if you specify an invalid index, such as 0 or a negative number.</td>
</tr>
<tr>
<td>InvalidVariableName</td>
<td>The policy schema validation failed because a variable name is invalid.</td>
</tr>
<tr>
<td>InvalidPayload</td>
<td>A payload specified in the policy is invalid.</td>
</tr>
</tbody>
</table>

Following fault variables are set when the policy triggers an error at runtime:

**Fault Variables**

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[prefix].[policy_name].failed</code></td>
<td>The <code>[prefix]</code> is assignmessage. The <code>[policy_name]</code> is the name of the policy that threw the error.</td>
<td>assignmessage.AM-SetResponse.failed = true</td>
</tr>
<tr>
<td><code>fault.name = [error_name]</code></td>
<td><code>[error_name]</code> is the specific error name to check for as listed in the table above.</td>
<td>fault.name = &quot;UnresolvedVariable&quot;</td>
</tr>
</tbody>
</table>

Following is an example of an error response:

```
{  
    "fault":{
        "detail":{
            "errorcode":"steps.assignmessage.VariableOfNonMsgType"
          ,
            "faultstring":"AssignMessage[AM-SetResponse]: value of variable is not of type Message"
        }
    }
}
```

Following is an example of a fault rule:

```
<faultrule name="VariableOfNonMsgType"></faultrule>
<FaultRule name="Assign Message Faults">
    <Step>
        <Name>AM-CustomNonMessageTypeErrorResponse</Name>
        <Condition>(fault.name Matches "VariableOfNonMsgType") </Condition>
    </Step>
</FaultRule>
```
Basic Authentication policy takes a username and password, Base64 encodes them, and writes the resulting value to a variable. The resulting value is typically written to an HTTP header, such as the Authorization header in the form Basic Base64EncodeString.

**Note**
This policy does not enforce Basic Authentication on a request to an API proxy. Instead, you use it to Base64 encode or decode credentials, typically when connecting to a backend server or using a service callout policy that requires Basic Authentication.

The policy has two modes of operations:
- **Encode**: Base64 encodes a username and password stored in variables
- **Decode**: Decodes the username and password from a Base64 encoded string

The username and password are commonly stored in the key/value store and then read from the key/value store at runtime.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>Request</th>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostFlow</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Flow</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>PreFlow</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<Name>AM-CustomSetVariableErrorResponse</Name>
<Condition>(fault.name = "SetVariableFailed")</Condition>
</Step>
<Condition>(assignmessage.failed = true) </Condition>
</FaultRule>
```

---

`Related Information`

Access Entity [page 99]

### 1.5.1.3.4.1.4 Basic Authentication

Basic Authentication policy takes a username and password, Base64 encodes them, and writes the resulting value to a variable. The resulting value is typically written to an HTTP header, such as the Authorization header in the form Basic Base64EncodeString.

**Note**
This policy does not enforce Basic Authentication on a request to an API proxy. Instead, you use it to Base64 encode or decode credentials, typically when connecting to a backend server or using a service callout policy that requires Basic Authentication.

The policy has two modes of operations:
- **Encode**: Base64 encodes a username and password stored in variables
- **Decode**: Decodes the username and password from a Base64 encoded string

The username and password are commonly stored in the key/value store and then read from the key/value store at runtime.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>Request</th>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostFlow</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Flow</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>PreFlow</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<Name>AM-CustomSetVariableErrorResponse</Name>
<Condition>(fault.name = "SetVariableFailed")</Condition>
</Step>
<Condition>(assignmessage.failed = true) </Condition>
</FaultRule>
```
<BasicAuthentication async='true' continueOnError='false' enabled='true'
xmlns='http://www.sap.com/apimgmt'>
   <Operation>Encode</Operation>
   <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
   <User ref='request.formparam.username'></User>
   <Password ref='request.formparam.password'></Password>
   <AssignTo createNew="false">request.header.Authorization</AssignTo>
</BasicAuthentication>

<!–- Use case: Extract the user credentials from the authorization header
User name and password is extracted from the authorization header of the
incoming request.
The user name and password is set into the flow variables named as
“current.username” and “current.password” respectively. -->
<BasicAuthentication async='true' continueOnError='false' enabled='true'
xmlns='http://www.sap.com/apimgmt'>
   <Operation>Decode</Operation>
   <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
   <User ref='current.username'></User>
   <Password ref='current.password'></Password>
   <Source>request.header.Authorization</Source>
</BasicAuthentication>

### Elements and Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation (Mandatory)</td>
<td>Supports values Encode or Decode. This setting will enable you to encode credentials to populate an HTTP header on an outbound request, or decode encoded credentials from HTTP header of an inbound request.</td>
</tr>
<tr>
<td>IgnoreUnresolvedVariables (Optional)</td>
<td>Supports values true or false. This setting determines whether to throw an error if the variables defined in the policy is not resolved. If set to true, the policy will not throw an error if a variable cannot be resolved. In Basic Authentication policy, it is recommended to set this value to false, because it is beneficial to throw an error if a username or password cannot be found in the variables specified.</td>
</tr>
<tr>
<td>User ref (Mandatory)</td>
<td>Settings for username. For encoding, set a reference attribute to the username to dynamically retrieve value from a variable. For decoding, specify the flow variable in which the decoded username is to be placed.</td>
</tr>
<tr>
<td>Password ref (Mandatory)</td>
<td>Settings for password. For encoding, set a reference attribute to the password to dynamically retrieve the value from a variable. For decoding, specify the flow variable in which the decoded password is to be placed.</td>
</tr>
<tr>
<td>AssignTo</td>
<td>Assigns the encoded value of username and password to a variable. Do not use this if the operation is Decode.</td>
</tr>
<tr>
<td>Source</td>
<td>The encoded value of username and password is retrieved from Source. Do not use this if the operation is Encode.</td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:
### Error Cause

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnresolvedVariable</td>
<td>The required source variables for the decode or encode are not present. This error can only occur if IgnoreUnresolved-Variables is false.</td>
</tr>
<tr>
<td>InvalidBasicAuthenticationSource</td>
<td>On a decode when the incoming Base64 encoded string does not contain a valid value or the header is malformed (for example does not start with “Basic”).</td>
</tr>
<tr>
<td>UserNameRequired</td>
<td>The &lt;User&gt; element must be present for the named operation. See the fault string.</td>
</tr>
<tr>
<td>PasswordRequired</td>
<td>The &lt;Password&gt; element must be present for the named operation. See the fault string.</td>
</tr>
<tr>
<td>AssignToRequired</td>
<td>The &lt;AssignTo&gt; element must be present for the named operation. See the fault string.</td>
</tr>
<tr>
<td>SourceRequired</td>
<td>The &lt;Source&gt; element must be present for the named operation. See the fault string.</td>
</tr>
</tbody>
</table>

Following fault variables are set when the policy triggers an error at runtime:

#### Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The [prefix] is BasicAuthentication.</td>
<td>BasicAuthentication.BA-Authenticate.failed = true</td>
</tr>
<tr>
<td></td>
<td>The [policy_name] is the name of the policy that threw the error.</td>
<td></td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name Matches &quot;UnresolvedVariable&quot;</td>
</tr>
</tbody>
</table>

### Related Information

**Key Value Map Operations [page 201]**

### 1.5.1.3.4.1.5 Concurrent Rate Limit

**Note:** The Concurrent Rate Limit policy is now deprecated.

The Concurrent Rate Limit policy was designed to be used with slow backend systems, however, the architecture used by it affected the performance of the APIs. Proxy chaining can be used. If you are experiencing slower performance with Concurrent Rate Limit policy, you can try using Spike Arrest instead. Using such features along with policy, will allow you to protect slow or sluggish backend systems without impacting the performance of the APIs. Migrate proxies using the Concurrent Rate Limit policy to the Quota policy or the Spike Arrest policies.
The Concurrent Rate Limit policy helps to limit the number of connections to your backend services from API proxies running on API Management.

Generally in a distributed environment, many API Proxies can point to the same backend service. In such a case, the backend service is flooded with requests. In order to manage traffic to backend services, you use the Concurrent Rate Limit policy. When the connection limit is exceeded, additional requests return HTTP response code 503: Service Unavailable. The ConcurrentRateLimit policy must be attached as a step to three Flows on a TargetEndpoint: request, response, and DefaultFaultRule. Also, when an API proxy is redeployed, the counter values are reset.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Request</td>
</tr>
<tr>
<td>PostClientFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>PreFlow</td>
<td>Flow</td>
</tr>
<tr>
<td>Flow</td>
<td>PostFlow</td>
</tr>
<tr>
<td>PostFlow</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response</td>
</tr>
<tr>
<td></td>
<td>PostClientFlow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<!-- The policy will limit the concurrent connections to target end point from API Runtime to 1. -->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ConcurrentRatelimit async="true" continueOnError="false" enabled="true"
 xmlns="http://www.sap.com/apimgmt">
  <AllowConnections count="1" ttl="10"/>
  <Distributed>true</Distributed>
  <StrictOnTtl>false</StrictOnTtl>
  <TargetIdentifier name="default"/>
</ConcurrentRatelimit>
```

```xml
<AllowConnections> element
```
<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowConnections (Required)</td>
<td>N/A</td>
<td>N/A</td>
<td>This element provides the number of concurrent connections between API Management and a back-end service that are allowed at any given time.</td>
</tr>
</tbody>
</table>

### Sample Code

```
<ConcurrentRateLimit async="true" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <AllowConnections count="1" ttl="10"/>
</ConcurrentRateLimit>
```

**count attribute of the `<AllowConnections>` element**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count (Required)</td>
<td>N/A</td>
<td>Integer</td>
<td>Specifies the number of concurrent connections between API Management and a back-end service that are allowed at any given time.</td>
</tr>
</tbody>
</table>

**ttl attribute of the `<AllowConnections>` element**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttl (Required)</td>
<td>N/A</td>
<td>Integer</td>
<td>The attribute <code>ttl</code> can be added to <code>&lt;AllowConnections&gt;</code> element to cause the counter to automatically decrement after the number of seconds configured. This can clean up any connections that were not decremented properly in the response path.</td>
</tr>
</tbody>
</table>

*Distributed* `element`
<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed (Optional)</td>
<td>false</td>
<td>Boolean</td>
<td>A Boolean that determines whether counter values are shared across the instances of API Management's server infrastructure.</td>
</tr>
</tbody>
</table>

**Sample Code**

```xml
<ConcurrentRateLimit
  async="true"
  continueOnError="false"
  enabled="true"
  xmlns="http://www.sap.com/apimgmt">
  <Distributed>true</Distributed>
</ConcurrentRateLimit>
```

**<StrictOnTtl> element**
### StrictOnTtl (Optional)

<table>
<thead>
<tr>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>Boolean</td>
<td>If set to true, the connection gets cut regardless of the back-end server throughput after the ttl is passed.</td>
</tr>
</tbody>
</table>

#### Sample Code

```
<ConcurrentRateLimit async="true" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <StrictOnTtl>false</StrictOnTtl>
</ConcurrentRateLimit>
```

### TargetIdentifier element

<table>
<thead>
<tr>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>The name of the Target Endpoint to which the limit should be applied.</td>
</tr>
</tbody>
</table>

#### Sample Code

```
<ConcurrentRateLimit async="true" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <TargetIdentifier name="default"/>
</ConcurrentRateLimit>
```
**name attribute of the <TargetIdentifier> element**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name (Optional)</td>
<td>N/A</td>
<td>N/A</td>
<td>Specifies the name of the Target Endpoint to which the limit should be applied.</td>
</tr>
</tbody>
</table>

A set of predefined flow variables are populated each time the policy executes:

- `concurrentratelimit.{policy_name}.allowed.count`
- `concurrentratelimit.{policy_name}.used.count`
- `concurrentratelimit.{policy_name}.available.count`
- `concurrentratelimit.{policy_name}.identifier`

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConcurrentRatelimitViolation</td>
<td>ConcurrentRatelimit connection exceeded. Connection limit : {0}</td>
</tr>
<tr>
<td>InvalidCountValue</td>
<td>ConcurrentRatelimit invalid count value specified.</td>
</tr>
<tr>
<td>ConcurrentRatelimitStepAttachment\</td>
<td>Concurrent Ratelimit policy (0) attachment is not allowed at proxy request/response/fault paths. This policy must be placed on the Target Endpoint.</td>
</tr>
<tr>
<td>NotAllowedAtProxyEndpoint</td>
<td>Concurrent Ratelimit policy (0) attachment is not allowed at proxy request/response/fault paths. This policy must be placed on the Target Endpoint.</td>
</tr>
<tr>
<td>MissingAtTargetEndpoint</td>
<td>Concurrent Ratelimit policy (0) attachment is missing at target request/response/fault paths. This policy must be placed in the Target Request Preflow, Target Response Postflow, and DefaultFaultRule.</td>
</tr>
<tr>
<td>InvalidTTLForMessageTimeOut</td>
<td>ConcurrentRatelimit invalid ttl value specified for message timeout. It must be a positive integer.</td>
</tr>
</tbody>
</table>

Following fault variables are set when the policy triggers an error at runtime:

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The fault variable [prefix] is concurrentratelimit. The [policy_name] is the name of the policy that threw the error.</td>
<td>concurrentratelimit.CRL-RateLimitPolicy.failed = true</td>
</tr>
<tr>
<td>fault [error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name Matches &quot;ConcurrentRateLimitViolation&quot;</td>
</tr>
</tbody>
</table>

**Related Information**

*Quota [page 217]*
1.5.1.3.4.1.6 Replace Concurrent Rate Limit Policy with Alternative Policies

The Concurrent Rate Limit policy was designed to cater to slow backend systems, however, the architecture used by it affected the performance of the APIs. Therefore, this policy has been deprecated.

**Note**

You can use the Spike Arrest policy to limit the number of requests to the backend systems over a specified period of time. You can also use Quota policy to limit the number of request messages that an API proxy allows over a period of time, such as minute, hour, day, week, or month. The Quota policy shouldn’t be used to protect target backend systems against traffic spikes.

If you’re wondering which policy to use to best meet your rate limiting needs, refer the following comparison chart:

<table>
<thead>
<tr>
<th>Quota</th>
<th>Spike Arrest</th>
<th>Concurrent Rate Limit (Deprecated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use it to limit the number of connections apps can make to your API proxy’s target backend over a specific period of time.</td>
<td>Use it to protect your API proxy’s target backend against severe traffic spikes and denial of service attacks.</td>
<td>Use it to limit the number of concurrent connections apps can make to your API proxy’s target backend.</td>
</tr>
<tr>
<td>Don’t use it to protect your API proxy’s target backend against traffic spikes. Use the Spike Arrest policy instead.</td>
<td>Don’t use it to count and limit the number of connections apps can make to your API proxy’s target backend over a specific period of time. Use the Quota policy instead.</td>
<td>Don’t use it to limit the number of connections applications can make to your API proxy’s target backend over a specific period of time. Use the Quota policy for this purpose.</td>
</tr>
<tr>
<td>The Quota policy stores the count.</td>
<td>The Spike Arrest policy doesn’t store the count.</td>
<td>The Concurrent Rate Limit policy stores the count.</td>
</tr>
</tbody>
</table>
| Attach the policy to the **ProxyEndpoint Request Pre-Flow**, generally after the authentication of the user. This enables the policy to check the quota counter at the entry point of your API proxy. | Attach the policy to the **ProxyEndpoint Request Pre-Flow**, generally at the very beginning of the flow. This provides spike protection at the entry point of your API proxy. | This policy must be attached in these three locations:  
- TargetEndpoint Request Pre-Flow  
- TargetEndpoint Response Pre-Flow  
- TargetEndpoint DefaultFaultRule |
<p>| HTTP status code when limit has been reached: 500 (Internal Server Error) * | HTTP status code when limit has been reached: 500 (Internal Server Error) * | HTTP status code when limit has been reached: 503 (Service Unavailable) |</p>
<table>
<thead>
<tr>
<th>Quota</th>
<th>Spike Arrest</th>
<th>Concurrent Rate Limit (Deprecated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good to know facts:</td>
<td>Good to know facts:</td>
<td>Good to know facts:</td>
</tr>
<tr>
<td>● Quota counter is stored in Cassandra.</td>
<td>● Performs throttling based on the time at which the last traffic was received. This time is stored per message processor.</td>
<td>● Keeps a count of concurrent connections per message processor.</td>
</tr>
<tr>
<td>● Configure the policy to synchronize the counter asynchronously to save resources.</td>
<td>● If you specify a rate limit of 100 calls per second, only 1 call every 1/100 second (10 ms) is allowed on the message processor. A second call within 10 ms is rejected.</td>
<td>● While an individual API proxy handles just a few connections collectively, the connections to a set of replicated API proxies pointing to the same backend service swamp the capacity of the service. Use this policy to limit this traffic to a manageable number of connections.</td>
</tr>
<tr>
<td>● Asynchronous counter synchronization can cause a delay in the rate limiting response, which allows calls slightly in excess of the limit you’ve set.</td>
<td>● Even with a high rate limit per second, nearly simultaneous requests results in rejections.</td>
<td>● This policy is known to slow performance in API proxies that handle a high number of transactions per second (TPS). For high-TPS API proxies, if ConcurrentRateLimit slows performance to unacceptable levels, try using SpikeArrest instead.</td>
</tr>
</tbody>
</table>

See [Quota](page 217) for more information.

See [Spike Arrest](page 234) for more information.

See [Concurrent Rate Limit](page 141) for more information.

### Example

Refer this example to replace Concurrent Rate Limit policy with Spike Arrest policy.

In Concurrent Rate Limit, 4 concurrent connections were allowed. In Spike Arrest, 4 requests per minute are allowed. You can alter the Spike Arrest Policy based on the number of requests allowed for your backend system.

### Sample Code

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SpikeArrest async="false" continueOnError="false" enabled="true" name="SpikeArrest-1">
  <DisplayName>Spike-Arrest-1</DisplayName>
  <Properties/>
  <UseEffectiveCount>true</UseEffectiveCount>
</SpikeArrest>
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ConcurrentRateLimit async="false" continueOnError="false" enabled="true" name="Concurrent Rate Limit-1">
  <DisplayName>Concurrent Rate Limit-1</DisplayName>
  <AllowConnections count="4" ttl="7"/>
  <Distributed>true</Distributed>
</ConcurrentRateLimit>
```
Concurrent Rate Limit and the Spike Arrest policy aren’t the same and works differently. However, you can tune it to allow specific requests per second, and set the maximum number of requests your backend system receives.

1.5.1.3.4.1.7 Extract Variables

The Extract variables policy can be used to extract content from the HTTP request or response messages of the API Proxy and assign that content to specific variables that can be accessed during the execution of the API Proxy.

For more information on how to extract different variables, see Examples [page 155].

This policy can be applied on the request or response stream of the proxy endpoint or target endpoint.

You can attach the policy in one of the following locations:

<table>
<thead>
<tr>
<th>Request</th>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td></td>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td></td>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<!-- The policy will extract data at xpath /Developer/Email and store in variable "email" for xml payload -->
<xml version="1.0" encoding="UTF-8" standalone="yes"/>
<ExtractVariables async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <Source>AccessEntity.GetDeveloperProfile</Source>
  <XMLPayload>
    <Variable name="email" type="string">
      <XPath>/Developer/Email</XPath>
    </Variable>
  </XMLPayload>
</ExtractVariables>
```
<table>
<thead>
<tr>
<th>Element</th>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>ignoreCase</td>
<td>Specifies the pattern to be applied to the parent element. The list of parent elements are: ● URIPath ● QueryParam ● FormParam ● Header ● Variable</td>
</tr>
<tr>
<td>&lt;URIPath&gt;</td>
<td></td>
<td>Extracts the value of a variable from the specified URI path of the specified message. This element is applicable only if source is request. * You must include at least one of the following: &lt;URIPath&gt;, &lt;QueryParam&gt;, &lt;Header&gt;, &lt;FormParam&gt;, &lt;JSONPayload&gt;, or &lt;XMLPayload&gt;</td>
</tr>
<tr>
<td>&lt;QueryParam&gt;</td>
<td>name (Mandatory)</td>
<td>Extracts the value of a variable from the specified query parameter of the specified message. * You must include at least one of the following: &lt;URIPath&gt;, &lt;QueryParam&gt;, &lt;Header&gt;, &lt;FormParam&gt;, &lt;JSONPayload&gt;, or &lt;XMLPayload&gt;</td>
</tr>
<tr>
<td>&lt;Header&gt;</td>
<td>name (Mandatory)</td>
<td>Extracts the value of a variable from the specified HTTP header of the specified message. * You must include at least one of the following: &lt;URIPath&gt;, &lt;QueryParam&gt;, &lt;Header&gt;, &lt;FormParam&gt;, &lt;JSONPayload&gt;, or &lt;XMLPayload&gt;</td>
</tr>
<tr>
<td>&lt;FormParam&gt;</td>
<td>name (Mandatory)</td>
<td>Extracts the value of a variable from the specified form parameter. Form parameters can be extracted only when the content Type of the specified message is application/x-wwwform-urlencoded. * You must include at least one of the following: &lt;URIPath&gt;, &lt;QueryParam&gt;, &lt;Header&gt;, &lt;FormParam&gt;, &lt;JSONPayload&gt;, or &lt;XMLPayload&gt;</td>
</tr>
<tr>
<td>Element</td>
<td>Attribute Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>&lt;Variable&gt;</code></td>
<td>name (Mandatory)</td>
<td>Specifies the name of the variable to which the extracted value will be assigned. If there is a VariablePrefix element, then this name will be appended, as variableprefix.name. For example, suppose the name is specified as Developer: If <code>&lt;VariablePrefix&gt;</code> is not specified, the extracted values are assigned to Developer. If <code>&lt;VariablePrefix&gt;</code> is specified as <code>MyUser</code>, the extracted values are assigned to <code>MyUser.Developer</code>.</td>
</tr>
<tr>
<td><code>&lt;IgnoreUnresolvedVariables&gt;</code></td>
<td></td>
<td>Set to <code>true</code> to treat any unresolvable variable as an empty string (null). Set to <code>false</code> if you want the policy to throw an error when any referenced variable is unresolvable.</td>
</tr>
</tbody>
</table>
| `<JSONPayload>`        |                              | Specifies the JSON formatted message from which the value of the variable will be extracted. *

You must include at least one of the following:

- `<URIPath>`
- `<QueryParam>`
- `<Header>`
- `<FormParam>`
- `<JSONPayload>`
- `<XMLPayload>` |
<p>| <code>&lt;JSONPayload&gt;</code>&lt;Variable&gt; |                              | Specific the variable where the extracted value will be assigned. |
| <code>&lt;JSONPayload&gt;</code>&lt;Variable&gt;&lt;JSONPath&gt; |                              | Specifies the JSON path used to extract the value of a variable from a JSON formatted message. JSON payloads are extracted only when the content-Type of the specified message is application/json. |</p>
<table>
<thead>
<tr>
<th>Element</th>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Source&gt;</code></td>
<td></td>
<td>Specifies the message to be parsed. The value of Source defaults to message. The message value is context-sensitive: In a request flow, message resolves to the request message. In a response flow, message resolves to the response message. If the source variable cannot be resolved, or resolves to a nonmessage type, the policy will fail to respond. In advanced configurations, source can resolve to a variable containing a message generated by another policy, such as one generated from a ServiceCallout.</td>
</tr>
<tr>
<td></td>
<td>clearPayload</td>
<td>Set to true is you want to clear the request payload after the request is sent to the HTTP target. Use the clearPayload option only if the request message is not required after the ServiceCallout is executed because clearPayload allocates memory during message processing.</td>
</tr>
<tr>
<td><code>&lt;VariablePrefix&gt;</code></td>
<td></td>
<td>The complete variable name is created by joining the <code>&lt;VariablePrefix&gt;</code>, a dot, and the name you define in curly braces in the Pattern element.</td>
</tr>
<tr>
<td><code>&lt;XMLPayload&gt;</code></td>
<td></td>
<td>Specifies the XML formatted message from which the value of the variable will be extracted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* You must include at least one of the following: <code>&lt;URIPath&gt;</code>, <code>&lt;QueryParam&gt;</code>, <code>&lt;Header&gt;</code>, <code>&lt;FormParam&gt;</code>, <code>&lt;JSONPayload&gt;</code>, or <code>&lt;XMLPayload&gt;</code></td>
</tr>
<tr>
<td></td>
<td>stopPayload</td>
<td>Set to true to stop XPath evaluation after one variable is populated.</td>
</tr>
<tr>
<td><code>&lt;XMLPayload&gt;</code></td>
<td><code>&lt;Namespaces&gt;</code></td>
<td>Specifies the namespace to be used in the XPath evaluation. XML payloads are extracted only when the contentType of the message is text/xml, application/xml, or application/*+xml</td>
</tr>
</tbody>
</table>

*Note*

When an XML variable is not resolved via an XPath expression, the ExtractVariables policy will result in an error. So, continueOnError or IgnoreUnresolvedVariables should be set to true to allow the execution of the policy.
## Element Description

<table>
<thead>
<tr>
<th>Element</th>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;XMLPayload&gt;</code>/ <code>&lt;Variable&gt;</code></td>
<td></td>
<td>Specifies variable to which the extracted value will be assigned.</td>
</tr>
<tr>
<td>type</td>
<td></td>
<td>Specifies the data type of the variable value. Supported data types are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- long</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- float</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- nodeset (returns an XML fragment)</td>
</tr>
<tr>
<td>XPath</td>
<td></td>
<td>Specifies the XPath defined for the variable. Only XPath 1.0 expressions are supported.</td>
</tr>
</tbody>
</table>

## During the policy execution, the following errors can occur:

### Error Cause

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecutionFailed</td>
<td>The policy tried to parse input that is malformed or otherwise invalid.</td>
</tr>
<tr>
<td></td>
<td>The policy tried to parse XML with a namespace that is not declared in the <code>&lt;Namespace&gt;</code> element.</td>
</tr>
<tr>
<td>SourceMessageNotAvailable</td>
<td>A variable specified in <code>&lt;Source&gt;</code> is out of scope or can’t be resolved. For example, if the policy executes in the request flow, the <code>&lt;Source&gt;</code> variable cannot be set to response or error.</td>
</tr>
<tr>
<td>SetVariableFailed</td>
<td>The policy was not able to set a variable value.</td>
</tr>
<tr>
<td>ImmutableVariable</td>
<td>A variable used in the policy is immutable. The policy was unable to set this variable.</td>
</tr>
<tr>
<td>VariableResolutionFailed</td>
<td>The policy could not resolve a variable. Be sure the variable you are trying to set exists in the runtime flow.</td>
</tr>
<tr>
<td>UnsupportedOperation</td>
<td>The policy tried to perform an unsupported operation on named flow variables.</td>
</tr>
<tr>
<td>UnableToCast</td>
<td>The policy was unable to cast a variable.</td>
</tr>
<tr>
<td>JSONPathCompilationFailed</td>
<td>The policy was unable to compile a JSON path expression.</td>
</tr>
<tr>
<td>JSONPathParsingFailure</td>
<td>The policy was unable to parse a JSON path to extract data into flow variables.</td>
</tr>
<tr>
<td>Error Name</td>
<td>Cause</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InvalidJSONPath</td>
<td>A JSON path used in the policy is invalid.</td>
</tr>
<tr>
<td>NothingToExtract</td>
<td>A required element is missing from the policy. At least one of these elements is required: URIPath, QueryParam, Header, FormParam, XMLPayload, JSONPayload.</td>
</tr>
<tr>
<td>NONEmptyPrefixMappedToEmptyURI</td>
<td>The &lt;XMLPayload&gt; element is not configured properly. A non-empty prefix cannot be mapped to an empty URI.</td>
</tr>
<tr>
<td>DuplicatePrefix</td>
<td>The &lt;XMLPayload&gt; element is not configured properly. There is a duplicate prefix.</td>
</tr>
<tr>
<td>NoXPathsToEvaluate</td>
<td>There are no XPaths to evaluate. An &lt;XPath&gt; child element must be specified.</td>
</tr>
<tr>
<td>EmptyXPathExpression</td>
<td>The policy has invalid configuration of the &lt;Variable&gt; child element of an &lt;XMLPayload&gt; element.</td>
</tr>
<tr>
<td>NoJSONPathsToEvaluate</td>
<td>You do not specify a &lt;JSONPath&gt; child element where it is required.</td>
</tr>
<tr>
<td>EmptyJSONPathExpression</td>
<td>The policy has an empty child element &lt;JSONPath&gt; in a element &lt;JSONPayload&gt;.</td>
</tr>
<tr>
<td>MissingName</td>
<td>The name attribute is missing from a policy element that requires it.</td>
</tr>
<tr>
<td>PatternWithoutVariable</td>
<td>A &lt;Pattern&gt; element that does not have a variable specified. The element requires the name of the variable in which extracted data will be stored.</td>
</tr>
<tr>
<td>CannotBeConvertedToNodeset</td>
<td>The result of an XPath expression cannot be converted to type nodeset.</td>
</tr>
<tr>
<td>JSONPathCompilationFailed</td>
<td>The policy could not compile a specified JSON Path.</td>
</tr>
<tr>
<td>InstantiationFailed</td>
<td>The policy could not be instantiated.</td>
</tr>
<tr>
<td>XPathCompilationFailed</td>
<td>The policy could not compile a specified XPath. Be sure to declare any namespaces that are used in the XPath.</td>
</tr>
<tr>
<td>InvalidPattern</td>
<td>A &lt;Pattern&gt; element is invalid in one of these elements: URIPath, QueryParam, Header, FormParam, XMLPayload, JSONPayload.</td>
</tr>
</tbody>
</table>

Following fault variables are set when the policy triggers an error at runtime:

Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The [prefix] is extractvariables. The [policy_name] is the name of the policy to check.</td>
<td>extractvariables.EV-ParseJsonResponse.failed = true</td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name = &quot;SourceMessageNotAvailable&quot;</td>
</tr>
</tbody>
</table>
1.5.1.3.4.1.7.1 Examples

You can use the Extract variables policy to choose the names of the variables to be set. You choose the source of the variable, and how many variables to extract and set. Below are some examples to illustrate how this policy works:

→ Remember

When an XML variable is not resolved via an XPath expression, the ExtractVariables policy will result in an error. So, continueOnError or IgnoreUnresolvedVariables should be set to true to allow the execution of the policy.

Extract a variable from a query parameter

Consider an example where your API design specifies that incoming requests must carry a query parameter named code, which holds a term that looks like ABCXXXX, where ABC is fixed, and the XXXXX denotes a varying string.

Sample Code

```xml
<ExtractVariables>
  <QueryParam name="code">
    <Pattern ignoreCase="true">ABC{abccode}</Pattern>
  </QueryParam>
  <VariablePrefix>queryinfo</VariablePrefix>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

Say, after a GET call, API Management sets the variable queryinfo.abccode to the value XXXXX. After API Management executes this Extract Variables policy, subsequent policies, or code in the processing flow can refer to the variable named queryinfo.abccode to get the string value XXXXX (for example, 12345).

Extract variables from a JSON payload

The Extract Variables policy can also extract values from JSON messages. The example below illustrates how to extract a variable from a portion of a JSON message payload. Consider this response payload:

Sample Code

```json
{
```
An element in the Extract Variables policy tells it to extract from a JSON payload. Rather than using text patterns, as you would when extracting values from headers, URI paths or query parameters, with JSON specify the portion to extract via a JSON path expression in which the $ character refers to the root node of the JSON. Here’s an example policy to illustrate:

**Sample Code**

```xml
<ExtractVariables>
  <Source>response</Source>
  <VariablePrefix>myresp</VariablePrefix>
  <JSONPayload>
    <Variable name="first_key">
      <JSONPath>$.results[0].key1</JSONPath>
    </Variable>
  </JSONPayload>
</ExtractVariables>
```

With this policy applied to the above message, API Management will extract a variable called `myresp.first_key` which will contain the value `value1`.

**Extract variables from an XML message**

You can extract variables from an XML payload, using XPath and explicitly named variables. Consider the below payload:

**Sample Code**

```xml
<RootElement>
  <Element1>
    <Element2 attr="myattr"/>
  </Element1>
</RootElement>
```

```xml
<ExtractVariables>
  <Source>response</Source>
  <VariablePrefix>myprefix</VariablePrefix>
  <XMLPayload>
    <Variable name="attrvalue" type="string">
      <XPath>/RootElement/Element1/Element2/@attr</XPath>
    </Variable>
  </XMLPayload>
</ExtractVariables>
```

With this policy, SAP API Management will set a variable called `myprefix.attrvalue` with the value `myattr`.

---

**SAP API Management**

SAP API Management in the Cloud Foundry Environment
### 1.5.1.3.4.1.8 Invalidate Cache

The cache can be invalidated explicitly by specifying an HTTP header. When a request that contains the specified HTTP header is received, the cache will be flushed.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>Preflow</td>
<td>Preflow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>Postflow</td>
<td>Postflow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<!-- The policy will clear the value of userId from the cache -->
<InvalidateCache async="false" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apимgmt">
  <CacheKey>
    <KeyFragment ref="request.header.userid"></KeyFragment>
  </CacheKey>
  <Scope>Exclusive</Scope>
  <PurgeChildEntries>true</PurgeChildEntries>
</InvalidateCache>
```

Invalidate Cache policy defines the following attributes that are common to all policy parent elements:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Default</th>
<th>Presence</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The internal name of the policy. Characters you can use in the name are</td>
<td>N/A</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>restricted to: A-Z 0-9_-$ %</td>
<td></td>
<td></td>
</tr>
<tr>
<td>continueOn-</td>
<td>Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>onError</td>
<td>false</td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>enabled</td>
<td>Set to true to enforce the policy. Set to false to &quot;turn off&quot; the policy. The policy will not be enforced even if it remains attached to a flow.</td>
<td>true</td>
<td>Optional</td>
</tr>
<tr>
<td>async</td>
<td>This attribute is deprecated.</td>
<td>false</td>
<td>Deprecated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheKey</td>
<td>Configures a unique pointer to a piece of data stored in the cache.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CacheResource</td>
<td>Specifies the cache where messages should be stored. A default cache is available.</td>
</tr>
<tr>
<td>Scope</td>
<td>Enumeration used to construct a prefix for a cache key when a Prefix element is not provided in the CacheKey element.</td>
</tr>
<tr>
<td>CacheContext</td>
<td>Specifies how to construct a cache key when a Prefix element value is not specified, or to clear cache entries added by another API proxy.</td>
</tr>
<tr>
<td>PurgeChildEntries</td>
<td>true to purge child cache entries when invalidating the cache. Default is false.</td>
</tr>
<tr>
<td>Prefix</td>
<td>Specifies a value to use as a cache key prefix.</td>
</tr>
<tr>
<td>KeyFragment</td>
<td>Specifies a value that should be included in the cache key, creating a namespace for matching requests to cached responses.</td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidCacheResourceReference</td>
<td>The cache specified in the &lt;CacheResource&gt; element does not exist.</td>
</tr>
<tr>
<td>CacheNotFound</td>
<td>The cache specified in the &lt;CacheResource&gt; element does not exist.</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.9 JavaScript

JavaScript Policy is used to configure the JavaScript code to execute within the context of an API proxy flow.

A JavaScript policy contains no actual code. Instead, a JavaScript policy references a JavaScript resource and defines the step in the API flow where the JavaScript executes. The JavaScript resource that is referenced by the JavaScript policy can be stored in the API Proxy bundle. The JavaScript resource always has a .js extension.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```
<!–- Use case: Remove forward slash from incoming URL present at the end using a library function in a different script file -->
<Javascript async="false" continueOnError="false" enabled="true"
timeLimit="200" xmlns='http://www.sap.com/apimgmt'>
  <IncludeURL>jsc://lib.js</IncludeURL>
```

SAP API Management in the Cloud Foundry Environment
```javascript
// Trims the last chars from the string
function trimEnd(value, searchChar) {
  if (value.charAt(value.length - 1) == searchChar) {
    value = value.substr(0, value.length - 1);
  }
  return value;
}
```

```javascript
// Returns the proxy path suffix by escaping the trailing '/'
function getTrimmedPathSuffix() {
  return trimEnd(context.getVariable("proxy.pathsuffix"), '/');
}
context.setVariable("trimmedPathSuffix", getTrimmedPathSuffix());
```

## Elements and Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeLimit (required)</td>
<td>Specifies the maximum time (in milliseconds) that the script is permitted to execute.</td>
</tr>
<tr>
<td>ResourceURL (required)</td>
<td>Specifies the JavaScript resource (file). This is the main code file from which the execution begins. Syntax: <code>&lt;ResourceURL&gt;jsc://example-javascript.js&lt;/ResourceURL&gt;</code></td>
</tr>
<tr>
<td>IncludeURL (optional)</td>
<td>Specifies a JavaScript library to be loaded as dependency. Store libraries under <code>/APIProxy/FileResources/</code>&lt;policy name &gt;.js in your API proxy bundle. The scripts are evaluated in the order in which they are listed in the policy. Syntax: <code>&lt;IncludeURL&gt;jsc://my-javascript-URL.js&lt;/IncludeURL&gt;</code></td>
</tr>
<tr>
<td>Display name (optional)</td>
<td>Labels the policy in the management UI proxy editor with a different, natural-language name. If you skip this element, the value of the <code>name</code> attribute is applied to the policy. Syntax: <code>&lt;DisplayName&gt;Policy-Display-Name&lt;/DisplayName&gt;</code></td>
</tr>
<tr>
<td>Property (optional)</td>
<td>Specifies a property that you can access from the JavaScript code at runtime. Syntax: <code>&lt;Properties&gt;&lt;Property name=&quot;propName&quot;&gt;property-value&lt;/Property&gt;&lt;/Properties&gt;</code></td>
</tr>
</tbody>
</table>
### SSL info (optional)

Specifies the properties that are used to configure TLS for all HTTP client instances created by the JavaScript policy.

**Syntax:**

```xml
<SSLInfo>
  <Enabled>trueFalse</Enabled>
  <ClientAuthEnabled>trueFalse</ClientAuthEnabled>
  <KeyStore>ref://keystoreRef</KeyStore>
  <KeyAlias>keyAlias</KeyAlias>
  <TrustStore>ref://truststoreRef</TrustStore>
</SSLInfo>
```

### During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScriptExecutionFailed</td>
<td>A runtime error occurred in the JavaScript code. See the fault string for details.</td>
</tr>
<tr>
<td>ScriptExecutionFailedLineNumber</td>
<td>An error occurred in the JavaScript code. See the fault string for details.</td>
</tr>
<tr>
<td>ScriptSecurityError</td>
<td>A security error occurred when the JavaScript executed. See the fault string for details.</td>
</tr>
<tr>
<td>WrongResourceType</td>
<td>In the <code>&lt;ResourceURL&gt;</code> and <code>&lt;IncludeURL&gt;</code> elements, you must refer to a JavaScript file correctly using the <code>js</code> resource type. For example, here is the correct way to refer to the JavaScript file in the policy: <code>&lt;ResourceURL&gt;jsc://JavaScript-1.js&lt;/ResourceURL&gt;</code></td>
</tr>
<tr>
<td>NoResourceForURL</td>
<td>The <code>&lt;ResourceURL&gt;</code> and <code>&lt;IncludeURL&gt;</code> elements refer to a JavaScript file that does not exist.</td>
</tr>
<tr>
<td>ScriptCompilationFailed</td>
<td>An error occurred during compilation of the JavaScript code. Refer to the error message for details.</td>
</tr>
<tr>
<td>InvalidResourceUrlFormat</td>
<td>This error occurs when the format of the resource URL specified within the <code>&lt;ResourceURL&gt;</code> or the <code>&lt;IncludeURL&gt;</code> element of the JavaScript policy is invalid, resulting in the deployment of the API proxy to fail.</td>
</tr>
<tr>
<td>InvalidResourceUrlReference</td>
<td>This error occurs when the <code>&lt;ResourceURL&gt;</code> or the <code>&lt;IncludeURL&gt;</code> elements refer to a JavaScript file that does not exist, resulting in the deployment of the API proxy to fail.</td>
</tr>
</tbody>
</table>

Following fault variables are set when the policy triggers an error at runtime:
Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The [prefix] is javascript. The [policy_name] is the name of the policy that threw the error.</td>
<td>javascript.JavaScript-1.failed = true</td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] is the specific error name to check for as listed in the table above.</td>
<td>fault.name Matches &quot;ScriptExecutionFailed&quot;</td>
</tr>
</tbody>
</table>

Following is an example of an error response:

```json
{
  "fault": {
    "faultstring": "Execution of SetResponse failed with error: Javascript runtime error: "ReferenceError: "status" is not defined. (setresponse.js: 6)"
  },
  "detail": {
    "errorcode": "steps.javascript.ScriptExecutionFailed"
  }
}
```

Following is an example of a fault rule:

```xml
<FaultRule name="JavaScript Policy Faults">
  <Step>
    <Name>AM-CustomErrorResponse</Name>
    <Condition>(fault.name Matches "ScriptExecutionFailed")</Condition>
  </Step>
  <Condition>(javascript.JavaScript.failed = true)</Condition>
</FaultRule>
```

# 1.5.1.3.4.1.10 Java Script Object Model

This topic describes JavaScript model for API Management. JavaScript model enables you to use the JavaScript policy to add custom JavaScript to an API proxy.

API Management JavaScript object model defines objects that can be used in a JavaScript code executing within an API proxy flow. Objects defined by the JavaScript object model are within the scope of the API proxy flow. On executing the JavaScript, a scope is created and within the scope, the following object references are created: `context`, `request`, `response`, and `crypto`. You can also use a `print` function for debugging purpose.

## Context Object

A context object is created for each request or response transaction executed by an API proxy. The context object exposes methods to get, set, and remove variables related to each transaction. Variables define properties specific to a transaction, and thus building logic that relies on these properties to execute custom behavior.
• **Scope**: Global; available throughout the API Proxy flow.

• **Child objects**: proxyRequest, proxyResponse, targetRequest, and targetResponse.

Child objects are scoped to the ambient request and response, either the proxy request and response or the target request and response. For example, if the JavaScript policy executes in the proxy endpoint part of the flow, then the context.proxyRequest and context.proxyResponse objects are in scope. If the JavaScript runs in a target flow, then the context.targetRequest and context.targetResponse objects are in scope.

Following table describes context objects and its children:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>A wrapper for the message processing pipeline context and the request and response Flows that are executed by the ProxyEndpoint and TargetEndpoint.</td>
<td>flow, session</td>
</tr>
<tr>
<td>context.proxyRequest</td>
<td>An object that represents the inbound request message to the ProxyEndpoint (from the requesting app to the API proxy)</td>
<td>headers, query parameters, method, body, url</td>
</tr>
<tr>
<td>context.targetRequest</td>
<td>An object that represents the outbound request message from the TargetEndpoint (from the API proxy to the backend service).</td>
<td>headers, query parameters, method, body, url</td>
</tr>
<tr>
<td>context.targetResponse</td>
<td>An object that represents the inbound target response message (from the backend service to the API proxy)</td>
<td>headers, content, status</td>
</tr>
<tr>
<td>context.proxyResponse</td>
<td>An object that represents the outbound proxy response message (from the API proxy to the requesting app)</td>
<td>headers, content, status</td>
</tr>
<tr>
<td>Context.flow</td>
<td>The name of the current flow.</td>
<td></td>
</tr>
<tr>
<td>Context.session</td>
<td>A map of name/value pairs that you can use to pass objects between two different steps executing in the same context. For example: context.session['key'] = 123.</td>
<td></td>
</tr>
</tbody>
</table>

**context.*Request child objects**

Each HTTP transaction executing in an API Proxy, creates two request message objects:

- Inbound: Request from client.
- Outbound: Request generated by API Proxy and submitted to the backend target.

**Tip**

You can use the object request to access the properties in a request flow. The request object refers to either context.proxyRequest or context.targetRequest, depending on where in the flow your JavaScript code executes.
context.*Request child object properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The url property is a read/write convenience property that combines scheme, host, port, path, and query parameters for the targetRequest. The complete URL of the request is composed of the following properties: protocol: The protocol of the URL (for example, HTTP, HTTPS) port: The port (for example: 80, 443) host: The host of the URL (for example, <a href="http://www.example.com">www.example.com</a>) path: The path of the URI (for example, /v1/mocktarget) When getting url, a URL is returned in the following format: protocol://host:port/path?queryParams Examples: context.targetRequest.url = '<a href="http://www.example.com/path?q1=1">http://www.example.com/path?q1=1</a>'; context.targetRequest.protocol = 'https';</td>
</tr>
<tr>
<td>headers</td>
<td>HTTP request headers as a mapping of String =&gt; List Examples: For this HTTP request: POST /v1/blogs HTTP/1.1 Host: api.example.com Content-Type: application/json Authorization: Bearer y1SkZIjbdWybfs4fUQe9BqP0LH5Z The following JavaScript: context.proxyRequest.headers['Content-Type']; context.proxyRequest.headers['Authorization']; returns the following values: application/json Bearer y1SkZIjbdWybfs4fUQe9BqP0LH5Z</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>method</td>
<td>The HTTP verb (GET, POST, PUT, DELETE, PATCH, etc.) associated with the request</td>
</tr>
</tbody>
</table>

Examples:

For this request:

```plaintext
POST /v1/blogs HTTP/1.1
Host: api.example.com
Content-Type: application/json
Authorization: Bearer ylSkZIjbdWybfs4fUQe9BqP0L952
```

The following JavaScript:

```javascript
context.proxyRequest.method;
```

returns the following value

```javascript
POST
```
### Property Name | Description
--- | ---
body | The message body (payload) of the HTTP request.

The request body has the following members:

context.targetRequest.body.asXML;

context.targetRequest.body.asJSON;

context.targetRequest.body.asForm;

Examples:

For an XML body:

```xml
<customer number='1'>
  <name>Fred</name>
<customer/>
```

To access the elements of the XML object as follows:

```javascript```
var name = context.targetRequest.body.asXML.name;
```

To access XML attributes, use the @ notation.

```javascript```
var number = context.targetRequest.body.asXML.@number;
```

For a JSON request body:

```json```
{
  "a": 1,
  "b": "2"
}
```

```javascript```
var a = context.proxyRequest.body.asJSON.a; // == 1
var b = context.proxyRequest.body.asJSON.b; // == 2
```

To read form parameters:

```javascript```
"vehicle=Car&vehicle=Truck"
```javascript```

v0 = context.proxyRequest.body.asForm['vehicle'][0];

v1 = context.proxyRequest.body.asForm['vehicle'][1];
```

### context.*Response child objects

Each HTTP transaction executing in an API Proxy, creates two response message objects:

- Inbound: Response from the backend service.
- Outbound: Response sent to client.

### Note

You can use the object `response` to access the properties in a request flow. The response object refers to either `context.proxyResponse` or `context.targetResponse`, depending on where in the flow your JavaScript code executes.
**Response object properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headers</td>
<td>The HTTP headers of the response message as a mapping of <code>String</code> =&gt; <code>List</code>.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td><code>var cookie = context.targetResponse.headers['Set-Cookie'];</code></td>
</tr>
<tr>
<td>status</td>
<td>The status code with status message as a property. Both status code and status message are available as properties.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td><code>var status = context.targetResponse.status.code; // 200</code></td>
</tr>
<tr>
<td></td>
<td><code>var msg = context.targetResponse.status.message; // &quot;OK&quot;</code></td>
</tr>
<tr>
<td>content</td>
<td>The HTTP body (payload content) of the response message.</td>
</tr>
<tr>
<td></td>
<td>Response content has the following members:</td>
</tr>
<tr>
<td></td>
<td><code>context.targetResponse.content.asXML;</code></td>
</tr>
<tr>
<td></td>
<td><code>context.targetResponse.content.asJSON;</code></td>
</tr>
</tbody>
</table>

**Context object methods**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>context.getVariable()</td>
<td>Retrieves the value of a predefined or custom variable.</td>
<td><code>context.getVariable(&quot;variable-name&quot;);</code></td>
</tr>
<tr>
<td>context.setVariable()</td>
<td>Sets the value for a custom variable or for any predefined variables.</td>
<td><code>context.setVariable(&quot;variable-name&quot;, value);</code></td>
</tr>
<tr>
<td>context.removeVariable()</td>
<td>Removes a variable from the context.</td>
<td><code>context.removeVariable('variable-name');</code></td>
</tr>
</tbody>
</table>

**Request and Response objects**

The request and response objects are "shorthand" references to the ambient request and response, either the proxy request and response or the target request and response. The objects these variables refer to depend upon the context in which the JavaScript policy executes. If the JavaScript runs in the flow of a proxy endpoint, then the request and response variables refer to `context.proxyRequest` and `context.ProxyResponse`. If the JavaScript runs in a target flow, then the variables refer to the `context.targetRequest` and `context.targetResponse`.

**Crypto Object**

Crypto object adds basic, high-performance cryptographic support to the JavaScript Object Model. Crypto object lets you perform basic cryptographic hashing functions in JavaScript.

- Scope: Global
- Hash objects supported by crypto:
  - SHA-1: You can create SHA-1 objects, update them, and convert them to hex and base64 values.
  - Create an SHA-1 object
var _sah1 = crypto.getSHA1();  
- Update an SHA-1 object  
  _sha512.update(value);  
- Return the SHA-1 object as a hex string  
  var _hashed_token = _sha1.digest();  
- Return the SHA-1 object as a base64 string  
  var _hashed_token = _sha1.digest64();

SHA-256: You can create SHA-256 objects, update them, and convert them to hex and base64 values.
- Create an SHA-256 object  
  var _sha256 = crypto.getSHA256();  
- Update an SHA-256 object  
  _sha256.update(value);  
- Return the SHA-256 object as a hex string  
  var _hashed_token = _sha256.digest();  
- Return the SHA-256 object as a base64 string  
  var _hashed_token = _sha256.digest64();

SHA-512: You can create SHA-512 objects, update them, and convert them to hex and base64 values.
- Create an SHA-512 object  
  var _sha512 = crypto.getSHA512();  
- Update an SHA-512 object  
  _sha512.update(value);  
- Return the SHA-512 object as a hex string  
  var _hashed_token = _sha512.digest();  
- Return the SHA-512 object as a base64 string  
  var _hashed_token = _sha512.digest64();

MD5: You can create MD5 objects, update them, and convert them to hex and base64 values.
- Create an MD5 object  
  var _md5 = crypto.getmd5();  
- Update an MD5 object  
  _md5 .update(value);  
- Return the MD5 object as a hex string  
  var _hashed_token = _md5.digest();  
- Return the MD5 object as a base64 string  
  var _hashed_token = _md5.digest64();

- **Crypto date/time support:** The crypto object supports date/time formatting patterns. 
  crypto.dateFormat() returns a date in the string format. 
  Syntax: 
  crypto.dateFormat(format, [timezone], [time]) 
  The following table shows the parameters and examples of Crypto date/time support:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>format (string)</td>
<td>The underlying implementation for this parameter is java.text.SimpleDateFormat, for example: 'YYYY-MM-DD HH:mm:ss.SSS'</td>
<td>Get the current time, down to milliseconds: var _now = crypto.dateFormat('YYYY-MM-DD HH:mm:ss.SSS');</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>timezone (string, optional)</td>
<td>The underlying implementation for this parameter is java.util.TimeZone. Default: UTC</td>
<td>Get the current time for Pacific Time Zone: var _pst = crypto.dateFormat('YYYY-MM-DD HH:mm:ss.SSS', 'PST');</td>
</tr>
<tr>
<td>time (number, optional)</td>
<td>A Unix timestamp value to format. Default: current time</td>
<td>Get the value of ten seconds from current time: var _timeNow = Number(context.getVariable('system.timestamp')); var ten_seconds = crypto.dateFormat('YYYY-MM-DD HH:mm:ss.SSS', 'PST', _timeNow + 10 * 1000);</td>
</tr>
</tbody>
</table>

### Sample with Crypto

```javascript
try {
  //get values to use with hash functions
  var salt = context.getVariable("salt") || 'SomeHardCodedSalt';
  var host = context.getVariable("request.header.Host");
  var _timeNow = Number(context.getVariable('system.timestamp'));
  var now = crypto.dateFormat('YYYY-MM-DD HH:mm:ss.SSS', 'PST', _timeNow);

  //generate a hash with the unhashedToken:
  var sha512 = crypto.getSHA512();
  sha512.update(salt);
  sha512.update(unhashed_token);
  //convert to base64
  var base64_token = sha512.digest64();

  // set headers
  context.setVariable("request.header.now", now);
  context.setVariable("request.header.token", base64_token);
} catch(e) {
  throw 'Error in Javascript';
}
```

### The print() function

If you are using the Java Script policy to execute custom Java Script code, then use the `print()` function to output debug information. Print function is directly available through JavaScript Object Model. For example:

```javascript
if (context.flow=="PROXY_REQ_FLOW") {
  print("In proxy request flow");
  var username = context.getVariable("request.queryparam.user");
  print("Got query param: " + username);
  context.setVariable("USER.name", username);
  print("Set query param: " + context.getVariable("USER.name");
}
if (context.flow=="TARGET_REQ_FLOW") {
  print("In target request flow");
  var username = context.getVariable("USER.name");
  var url = "http://www.abc.com/user?"
  context.setVariable("target.url", url + "user=" + username);
  print("callout to URL: ", context.getVariable("target.url");
```
Using http Client

The http Client object is exposed to custom Java Script code through the JavaScript object model. To attach custom JavaScript to an API proxy, you use the Java Script policy. When the policy runs, the custom JavaScript code executes.

The httpClient object is useful for developing composite services or mashups. For example, you can consolidate multiple backend calls into a single API method. This object is commonly used as an alternative to the Service Callout policy.

Here’s a basic usage pattern. Instantiate a Request object, assign to it a URL (e.g., to a backend service you want to call), and call httpClient.send with that request object.

```javascript
var myRequest = new Request();
myRequest.url = "http://www.abc.com";
var exchangeObj = httpClient.send(myRequest);
```

httpClient Reference

HTTP Client exposes two methods: get() and send()

- httpClient.get()
  
  **Usage:**
  ```javascript
  var exchangeObj = httpClient.get(url);
  ```
  
  **Return:** method returns an exchange object. This object has no properties, and it exposes the following methods:
  - **isError():** (boolean) Returns true if the httpClient was unable to connect to the server. HTTP status codes 4xx and 5xx result in isError() false, as the connection completed and a valid response code was returned. If isError() returns true, then a call to getResponse() returns the JavaScript undefined.
  - **isSuccess():** (boolean) Returns true if the send was complete and successful.
  - **isComplete():** (boolean) Returns true if the request is complete.
  - **waitForComplete():** Pauses the thread until the request is complete (by success or error).
  - **getResponse():** (object) Returns the response object if the httpClient.send() was complete and successful.
  - **getError():** (string) If the call to httpClient.send() resulted in an error, returns the error message as a string.

  You can use the exchange object later to get the actual HTTP response, or to check whether the response has timed out. For example:

  ```javascript
  var ex1 = httpClient.get("http://www.example.com?api1");
  context.session["ex1"] = ex1;  // Put the object into the session
  var ex2 = httpClient.get("http://www.example.com?api2");
  context.session["ex2"] = ex2;  // Put the object into the session
  ```

- httpClient.send()
  
  **Usage:**
  ```javascript
  var request = new Request(url, operation, headers);
  var exchangeObj = httpClient.send(request);
  ```
1.5.1.3.4.1.11 JSON to XML

API Management enables developers to convert messages from the JavaScript object notation (JSON) format to the extensible markup language (XML) format by using the JSON to XML policy type.

This policy is useful for enabling backend XML services to support RESTful apps that require JSON (for example, due to a lack of an XML parsing capabilities on the client).

In a typical mediation scenario, a JSON to XML policy on the inbound request flow is often paired with an XML to JSON policy on the outbound response flow. By combining policies this way, a JSON API can be exposed for services that natively support only XML.

For scenarios where APIs are consumed by diverse consumer applications which may require either JSON or XML, the format of the response can be dynamically set by configuring JSON to XML and XML to JSON policies to execute conditionally.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
<JSONToXML async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <Options>
    <ArrayItemElementName>item</ArrayItemElementName>
    <ArrayRootElementName>rootelement</ArrayRootElementName>
    <ObjectRootElementName>objectroot</ObjectRootElementName>
    <InvalidCharsReplacement/>
    <AttributePrefix/>
    <AttributeBlockName/>
    <TextNodeName/>
    <NamespaceSeparator/>
    <DefaultNamespaceNodeName/>
    <NullValue>I_AM_Null</NullValue>
  </Options>
  <OutputVariable>response</OutputVariable>
  <Source>response</Source>
</JSONToXML>
```
Following table lists the elements and attributes that you can configure on this policy

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source (optional)</td>
<td>The variable containing the JSON-formatted message to be converted to XML. Usually, you set this to request or response, depending on whether the message to be transformed is inbound or outbound.</td>
</tr>
<tr>
<td></td>
<td>If you have not defined the source, then it is treated as message, resolving to a request when the policy is attached to a request flow, or a response when the policy is attached to a response flow.</td>
</tr>
<tr>
<td></td>
<td>If the source variable cannot be resolved, or resolves to a non-message type, the policy throws an error.</td>
</tr>
<tr>
<td>OutputVariable</td>
<td>The variable name of the resultant XML-formatted message. Usually, you set this to request or response, depending on whether the message to be transformed is inbound or outbound. In most cases, a JSON request is transformed into an XML request. Similarly, a JSON response is usually transformed into an XML response.</td>
</tr>
<tr>
<td></td>
<td>If OutputVariable is not specified, then the source variable is treated as OutputVariable. That is, the policy assumes that a JSON request, for example, is being converted into an XML request.</td>
</tr>
<tr>
<td>i Note</td>
<td>This element is required when the variable defined in the Source element is a string.</td>
</tr>
<tr>
<td>InvalidCharsReplacement</td>
<td>To assist with handling invalid XML that may cause issues with a parser, this setting replaces any JSON elements that produce invalid XML with the string.</td>
</tr>
<tr>
<td></td>
<td>For example, the following setting:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;InvalidCharsReplacement&gt;_&lt;&lt;/InvalidCharsReplacement&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Converts this JSON object:</td>
</tr>
<tr>
<td></td>
<td><code>{ &quot;First%%Name&quot;: &quot;Adam&quot; }</code>                                                                -spacing:20px;overflow:hidden;user-select:all;white-space:pre-wrap;overflow-wrap:break-word;box-sizing:border-box;word-break:break-word;cursor:pointer;`</td>
</tr>
<tr>
<td></td>
<td>to this XML structure: <code>&lt;First_Name&gt;Adam&lt;First_Name&gt;</code></td>
</tr>
<tr>
<td><strong>Elements and Attributes</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>TextNodeName</td>
<td>Converts a JSON property into an XML text node with the specified name. For example, the following setting: <code>&lt;TextNodeName&gt;age&lt;/TextNodeName&gt;</code> converts this JSON:</td>
</tr>
<tr>
<td></td>
<td>```{</td>
</tr>
<tr>
<td></td>
<td>&quot;person&quot;: {</td>
</tr>
<tr>
<td></td>
<td>&quot;firstName&quot;: &quot;Adam&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;lastName&quot;: &quot;Philip&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;age&quot;: 30</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>to this XML structure:&lt;person&gt;&lt;firstName&gt;Adam&lt;/firstName&gt;&lt;age&gt;30&lt;/age&gt;&lt;lastName&gt;Philip&lt;/lastName&gt;&lt;/person&gt;</td>
</tr>
<tr>
<td></td>
<td>If TextNodeName is not specified, the XML is generated, using the default setting for a text node:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;person&gt;&lt;firstName&gt;Adam&lt;/firstName&gt;&lt;age&gt;30&lt;/age&gt;&lt;lastName&gt;Philip&lt;/lastName&gt;&lt;/person&gt;</code></td>
</tr>
<tr>
<td>NullValue</td>
<td>Indicates a null value. By default the value is NULL. For example the following setting: <code>&lt;NullValue&gt;NULL_VALUE&lt;/NullValue&gt;</code> converts the following JSON object: <code>{&quot;person&quot;: &quot;NULL_VALUE&quot;}</code> to the following XML element: <code>&lt;person&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Where no value (or a value other than NULL_VALUE) is specified for the Null value, then the same payload converts to: <code>&lt;person&gt;NULL_VALUE&lt;/person&gt;</code></td>
</tr>
<tr>
<td>Elements and Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| AttributeBlockName      | Enables you to specify when JSON elements are converted into XML attributes (rather than XML elements). For example, the following setting converts properties inside an object named #attrs into XML attributes: `<AttributeBlockName>#attrs</AttributeBlockName>` The following JSON object:

```json
{
    "person" : {
        "#attrs" : {
            "firstName" : "Adam",
            "lastName" : "Philip"
        },
        "location" : "California"
    }
}
```

is converted to the following XML structure:

```xml
<person firstName="Adam" lastName="Philip"><location>California</location></person>
```

| AttributePrefix         | Converts the property starting with the specified prefix into XML attributes. Where the attribute prefix is set to #, for example: `<AttributePrefix>#</AttributePrefix>` Converts the following JSON object:

```json
{
    "person" : {
        "#firstName" : "Adam",
        "#lastName" : "Philip",
        "location" : "California"
    }
}
```

```xml
<person firstName="Adam" lastName="Philip"><location>California</location></person>
```
Elements and Attributes | Description
--- | ---
NamespaceBlockName | JSON has no support for namespaces, while XML documents often require them. The NamespaceBlockName enables you to define a JSON property that serves as the source of a namespace definition in the XML that is produced by the policy. (This means that the source JSON must provide a property that can be mapped into a namespace that is expected by application that consumes the resulting XML.)
DefaultNamespaceNodeName
NameSpaceSeparator

For example, the following settings:

```xml
<NamespaceBlockName>#namespaceblock</NamespaceBlockName>
<DefaultNamespaceNodeName>$defaultname</DefaultNamespaceNodeName>
<NameSpaceSeparator>:</NameSpaceSeparator>
```

indicates that a property called #namespaceblock exists in the source JSON that contains at least one namespace designated as the default. For example:

```json
{
    "vehicle": {
        "#namespaceblock": {
            "$default": "http://www.w3.org/1999/name",
            "xmlns:model": "http://www.w3.org/1999/models"
        },
        "name": "Car",
        "models:name": "Alto"
    }
}
```

converts to:

```xml
<name xmlns="http://www.w3.org/1999/name" xmlns:exp="http://www.w3.org/1999/models">
    <name>Car</name>
    <models:name>Alto</models:name>
</name>
```
<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayRootElementName</td>
<td>Converts a JSON array into a list of XML elements with specified parent and child element names.</td>
</tr>
<tr>
<td>ArrayItemElementName</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For example, the following settings:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;ArrayRootElementName&gt;Array&lt;/ArrayRootElementName&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;ArrayItemElementName&gt;Item&lt;/ArrayItemElementName&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Converts the following JSON array:</td>
</tr>
<tr>
<td></td>
<td>`[ &quot;Doctor&quot;,</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;occupation&quot;: &quot;Engineer&quot;</td>
</tr>
<tr>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td>&quot;Scientist&quot;</td>
</tr>
<tr>
<td></td>
<td>]`</td>
</tr>
<tr>
<td></td>
<td>into the following XML structure:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Array&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Item&gt;Dcotor&lt;/Item&gt;</code></td>
</tr>
<tr>
<td></td>
<td>`&lt;Item&gt;</td>
</tr>
<tr>
<td></td>
<td><code>&lt;occupation&gt;Engineer&lt;/occupation&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/Item&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Item&gt;Scientist&lt;/Item&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/Array&gt;</code></td>
</tr>
<tr>
<td>ObjectRootElementName</td>
<td>Specifies the root element name when you convert from JSON, which does not have a named root element, to XML.</td>
</tr>
<tr>
<td></td>
<td>For example, if the JSON appears as:</td>
</tr>
<tr>
<td></td>
<td>`{ &quot;xyz&quot;: &quot;123&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;abc&quot;: &quot;234&quot;</td>
</tr>
<tr>
<td></td>
<td>}`</td>
</tr>
<tr>
<td></td>
<td>And you set the ObjectRootElementName as:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;ObjectRootElementName&gt;Root&lt;/ObjectRootElementName&gt;</code></td>
</tr>
<tr>
<td></td>
<td>The resulting XML appears as:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Root&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xyz&gt;123&lt;/xyz&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;abc&gt;234&lt;/abc&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/Root&gt;</code></td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:
<table>
<thead>
<tr>
<th>Error Cause</th>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SourceUnavailable</td>
<td>The variable specified in the Source element is not available or cannot be resolved.</td>
</tr>
<tr>
<td></td>
<td>ExecutionFailed</td>
<td>The input payload (JSON) is empty or the input (JSON) passed to XML policy is invalid or malformed.</td>
</tr>
<tr>
<td></td>
<td>OutputVariableIsNotAvailable</td>
<td>The variable specified in the Source element of the JSON to XML Policy is of type string and the OutputVariable is not defined.</td>
</tr>
<tr>
<td></td>
<td>InCompatibleTypes</td>
<td>The type of the variable defined in the Source and OutputVariable element does not match. The valid types are message and string.</td>
</tr>
<tr>
<td></td>
<td>InvalidSourceType</td>
<td>The type of the variable used to define the Source element is invalid. The valid types are message and string.</td>
</tr>
</tbody>
</table>

The following fault variables are set when the policy triggers an error at runtime:

**Fault Variables**

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The variable [prefix] is jsontoxml. The [policy_name] is the name of the policy that threw the error.</td>
<td>jsontoxml.JSON-to-XML-1.failed = true</td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name Matches &quot;SourceUnavailable&quot;</td>
</tr>
</tbody>
</table>

Following is an example of an error response:

```
{
  "fault": {
    "faultstring": "JSONToXML[JSON-to-XML-1]: Source abc is not available",
    "detail": {
      "errorcode": "steps.json2xml.SourceUnavailable"
    }
  }
}
```

Following is an example of a fault rule:

```
<FaultRule name="JSON To XML Faults">
  <Step>
    <Name>AM-SourceUnavailableMessage</Name>
    <Condition>(fault.name Matches "SourceUnavailable")</Condition>
  </Step>
  <Step>
    <Name>AM-BadJSON</Name>
    <Condition>(fault.name = "ExecutionFailed")</Condition>
  </Step>
  <Condition>(jsontoxml.JSON-to-XML-1.failed = true)</Condition>
</FaultRule>
```
Related Information

XML to JSON [page 270]
XSL Transform [page 281]

1.5.1.3.4.1.12 JSON Threat Protection

Minimizes the risk posed by content-level attacks by enabling specific limits on various JSON structures, such as arrays and strings.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>Proxy Endpoint</th>
<th>Target Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

Note

JSON Threat Protection policy can be applied only to POST or PUT operations of an API. Applying this policy to a GET operation of an API results in an error.

An example payload for the policy is as follows:

```xml
<JSONThreatProtection async="false" continueOnError="false" enabled="true"
    xmlns="http://www.sap.com/apimgmt">
    <ArrayElementCount>15</ArrayElementCount>
    <ContainerDepth>15</ContainerDepth>
    <ObjectEntryCount>15</ObjectEntryCount>
    <ObjectEntryNameLength>25</ObjectEntryNameLength>
    <Source>request</Source>
    <StringValueLength>100</StringValueLength>
</JSONThreatProtection>
```
<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Element Count (Optional)</td>
<td>This attribute specifies the maximum number of elements allowed in an array.</td>
</tr>
<tr>
<td></td>
<td>The limit is not applied if you do not specify this element, or if you specify a negative integer.</td>
</tr>
<tr>
<td></td>
<td>Syntax: <code>&lt;ArrayElementCount&gt;10&lt;/ArrayElementCount&gt;</code></td>
</tr>
<tr>
<td>Container Depth (Optional)</td>
<td>This attribute specifies the maximum container depth allowed for objects or arrays.</td>
</tr>
<tr>
<td></td>
<td>For example, the container depth of an array containing an object, which contains another object is 3.</td>
</tr>
<tr>
<td></td>
<td>The limit is not applied if you do not specify this element, or if you specify a negative integer.</td>
</tr>
<tr>
<td></td>
<td>Syntax: <code>&lt;ContainerDepth&gt;5&lt;/ContainerDepth&gt;</code></td>
</tr>
<tr>
<td>Object Entry Count (Optional)</td>
<td>This attribute specifies the maximum number of entries allowed within an object.</td>
</tr>
<tr>
<td></td>
<td>The limit is not applied if you do not specify this element, or if you specify a negative integer.</td>
</tr>
<tr>
<td></td>
<td>Syntax: <code>&lt;ObjectEntryCount&gt;10&lt;/ObjectEntryCount&gt;</code></td>
</tr>
<tr>
<td>Object Entry Length Name (Optional)</td>
<td>This attribute specifies the maximum string length allowed for a property name within an object.</td>
</tr>
<tr>
<td></td>
<td>The limit is not applied if you do not specify this element, or if you specify a negative integer.</td>
</tr>
<tr>
<td></td>
<td>Syntax: <code>&lt;ObjectEntryNameLength&gt;100&lt;/ObjectEntryNameLength&gt;</code></td>
</tr>
<tr>
<td>Source (Optional)</td>
<td>This attribute indicates the message to be screened for JSON payload attacks.</td>
</tr>
<tr>
<td></td>
<td><strong>Request:</strong> With a threat protection policy attached to any request flow, invalid messages return a 400 status code, along with a corresponding policy error message.</td>
</tr>
<tr>
<td></td>
<td><strong>Response:</strong> Threat protection policy attached to any response flow, invalid messages still returns a 500 status code, and one of the corresponding policy error messages is thrown (rather than just ExecutionFailed).</td>
</tr>
<tr>
<td></td>
<td>Syntax: <code>&lt;Source&gt;response&lt;/Source&gt;</code></td>
</tr>
</tbody>
</table>
Elements and Attributes

Description

String Value Length (Optional)

This attribute specifies the maximum length allowed for a string value.

The limit is not applied if you do not specify this element, or if you specify a negative integer.

Syntax: `<StringValueLength>200</StringValueLength>`

During the policy execution, the following errors can occur:

**Error Code**

<table>
<thead>
<tr>
<th>Error Name</th>
<th>HTTP Status</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExceededContainerDepth</td>
<td>500</td>
<td>JSONThreatProtection[policy_name]: Exceeded container depth at line [line_num]</td>
</tr>
<tr>
<td>ExceededObjectEntryCount</td>
<td>500</td>
<td>JSONThreatProtection[policy_name]: Exceeded object entry count at line [line_num]</td>
</tr>
<tr>
<td>ExceededArrayElementCount</td>
<td>500</td>
<td>JSONThreatProtection[policy_name]: Exceeded array element count at line [line_num]</td>
</tr>
<tr>
<td>ExceededObjectEntryNameLength</td>
<td>500</td>
<td>JSONThreatProtection[policy_name]: Exceeded object entry name length at line [line_num]</td>
</tr>
<tr>
<td>ExceededStringValueLength</td>
<td>500</td>
<td>JSONThreatProtection[policy_name]: Exceeded string value length at line [line_num]</td>
</tr>
<tr>
<td>Invalid JSON object</td>
<td>500</td>
<td>JSONThreatProtection[policy_name]: The input JSON Payload is invalid.</td>
</tr>
<tr>
<td>SourceUnavailable</td>
<td>500</td>
<td>This error occurs when the message variable mentioned in the source element is either unavailable in the specific flow where the policy is being executed or it does not have a valid value (request, response, or message). JSONThreatProtection[policy_name]: Source [var_name] is not available</td>
</tr>
<tr>
<td>NonMessageVariable</td>
<td>500</td>
<td>This error occurs when the source element is set to a variable type which is not a message. JSONThreatProtection[policy_name]: Variable [var_name] does not resolve to a Message</td>
</tr>
<tr>
<td>ExecutionFailed</td>
<td>500</td>
<td>JSONThreatProtection[policy_name]: Execution failed. reason: [string]</td>
</tr>
</tbody>
</table>

Following fault variables is set when the policy triggers an error at runtime:
### Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>[prefix]: jsonattack</td>
<td>jsonthreatprotection.JTP-SecureRequest.failed = true</td>
</tr>
<tr>
<td></td>
<td>[policy_name]: The name of the policy to check.</td>
<td></td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name Matches &quot;SourceUnavailable&quot;</td>
</tr>
</tbody>
</table>

### Following is an example of an error response:

```json
{
    "fault": {
        "faultstring": "JSONThreatProtection[JPT-SecureRequest]: Execution failed. reason: JSONThreatProtection[JTP-SecureRequest]: Exceeded object entry name length at line 5",
        "detail": {
            "errorcode": "steps.jsonthreatprotection.ExecutionFailed"
        }
    }
}
```

### Following is an example of a fault rule:

```xml
<FaultRule name="JSON Threat Protection Policy Faults">
    <Step>
        <Name>CustomErrorResponse</Name>
        <Condition>(fault.name Matches "ExecutionFailed") </Condition>
    </Step>
    <Condition>(jsonattack.JTP-SecureRequest.failed = true) </Condition>
</FaultRule>
```

### Related Information

- JSON to XML [page 170]
- XML Threat Protection [page 283]
- Regular Expression Protection [page 296]

### 1.5.1.3.4.1.13 JSON Web Tokens

This topic describes about JSON Web Tokens (JWT) policies available in API Management.

JSON Web Tokens, or JWT, are commonly used to share claims or assertions between connected applications. The JWT policies enable API proxies to:
- Generate signed JWTs.
- Verify digitally signed JWTs and claims within those JWTs.
- Decode signed JWTs without validating signatures on the token.

**Note**

If this policy is not visible, then it is yet to be enabled for your data center and it will be enabled shortly.

### Algorithms used in JWT policy

- HMAC algorithm: The HMAC algorithm uses a secret key for creating and verifying signatures.
- RSA algorithm: The RSA algorithm uses a public/private key pair for the cryptographic signature. With RSA signatures, the signing party uses a private key to sign the JWT, and the verifying party uses the matching public key to verify the signature on the JWT.

### Parts of a JWT

A signed JWT encodes information in three parts:

- Header
- Payload
- Signature

Generate JWT policy creates all the parts. Verify JWT policy examines all the parts, and Decode JWT policy examines the header and payload.

### JSON Web Key Set (JWKS)

A JSON Web Key Set (JWKS) is a JSON structure that represents a set of JSON Web Keys. A JSON Web Key (JWK) is a JSON data structure that represents a cryptographic key.

JWKS structure: Each key element in the JWKS must include these attributes.

- kty - Must be set to RSA.
- kid (the key id) - Arbitrary value (no duplicates within a key set). If the inbound JWT bears a key ID, which is present in the set of JWKS, then the policy will use the correct public key to verify the JWT signature.
- n - The RSA key value "modulus".
- e - The RSA key value "exponent".

Following are optional elements and their required values:

- alg - If present, must be RS256.
- use - If present, must be sig.

The following JWKS includes the required elements and values

**Sample Code**

```json
{
  "keys": [
    {
      "kty": "RSA",
      "alg": "RS256",
      "use": "sig",
      "kid": "ca04df587b5a7cead80abec9ea8d75f87a78e01",
      "n": "iXn-WmrwLBA-QbiToBozpu4Y4ThKdWORWFXQa9175pKovPUjUj2Bk05TUST7-V7KdJcq0_Nkd-
X9rMKV5LRqCa0_F8Yg130QS3bUm9orFryrdOc65PUVFVxIwMZuGy1hj6HEJWV10OCZdrgN1l06B
```
1.5.1.3.4.1.13.1 Generate JWT

This topic describes about Generate JSON Web Token (JWT) Policy.

This policy generates a signed JWT, with a configurable set of claims. The JWT can then be returned to clients, transmitted to backend targets, or used in other ways.
You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preflow</td>
<td>Preflow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

Generate a JWT signed with the HS256 algorithm

The sample provided below generates a new JWT and signs it using the HS256 algorithm. HS256 relies on a shared secret for both signing and verifying the signature.

```xml
<!-- Generate JWT policy -->
<GenerateJWT async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgt">
  <Algorithm>HS256</Algorithm>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <SecretKey>
    <Value ref="private.secretkey"/>
    <Id>1918290</Id>
  </SecretKey>
  <ExpiresIn>1h</ExpiresIn>
  <Subject>subject-subject</Subject>
  <Issuer>urn://apim-JWT-policy-test</Issuer>
  <Audience>audience1,audience2</Audience>
  <Id/>
  <AdditionalClaims>
    <Claim name="additional-claim-name" type="string">additional-claim-value-goes-here</Claim>
  </AdditionalClaims>
  <OutputVariable>jwt-variable</OutputVariable>
</GenerateJWT>
```

The resulting JWT will have this header and payload:

```json
# header
{
  "typ": "JWT",
  "alg": "HS256",
}

# payload
{
  "sub": "subject-subject",
  "iss": "urn://apim-JWT-policy-test",
  "aud": "additional-claim-name",
  "iat": 1506553019,
  "exp": 1506556619,
  "jti": "BD1FF263-3D25-4593-A685-5EC1326E1F37",
  "additional-claim-name": "additional-claim-value-goes-here"
}
```
Generate a JWT signed with the RS256 algorithm.

This example policy generates a new JWT and signs it using the RS256 algorithm. Generating an RS256 signature relies on an RSA private key, which must be provided in PEM-encoded form. When this policy action is triggered, Edge encodes and digitally signs the JWT, including the claims. To learn about the parts of a JWT and how they’re encrypted and signed, refer to RFC7519.

```xml
<!-- The policy -->
<GenerateJWT async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <Algorithm>RS256</Algorithm>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <PrivateKey>
    <Value ref="private.privatekey"/>
    <Password ref="private.privatekey-password"/>
    <Id ref="private.privatekey-id"/>
  </PrivateKey>
  <ExpiresIn>1h</ExpiresIn>
  <Subject>subject-subject</Subject>
  <Issuer>urn://apim-JWT-policy-test</Issuer>
  <Audience>audience1,audience2</Audience>
  <Id/>
  <AdditionalClaims>
    <Claim name="additional-claim-name" type="string">additional-claim-value-goes-here</Claim>
  </AdditionalClaims>
  <OutputVariable>jwt-variable</OutputVariable>
</GenerateJWT>
```

Following table lists the elements and attributes that you can configure on this policy.

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Specifies the encryption algorithm to sign the token. HS256 employs a shared secret. RS256 employs a public/secret key pair. Supported value: HS256, HS384, HS512, RS256, RS384, RS512</td>
</tr>
<tr>
<td></td>
<td>`&lt;Algorithm&gt;HS256</td>
</tr>
<tr>
<td>Audience (optional)</td>
<td>The policy generates a JWT containing an aud claim set to the specified value. This claim identifies the recipients that the JWT is intended for.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Audience&gt;audience&lt;/Audience&gt;</code></td>
</tr>
</tbody>
</table>
### AdditionalClaims (optional)

Specify additional claims in the payload of the JWT. You can specify the claim explicitly as string, a number, a boolean, a map, or an array. A map is simply a set of name/value pairs.

```xml
<AdditionalClaims>
  <Claim name='claim1'>explicit-value-of-claim-here</Claim>
  <Claim name='claim2' ref='var-name'/>
  <Claim name='claim3' ref='var-name' type='boolean'/>
</AdditionalClaims>
```

The `<Claim>` element takes these attributes:

- **name**: name of the claim.

  - **Note**: Don’t use any of the registered claim names in this element. They include: “iss”, “sub”, “aud”, “iat”, “exp”, “nbf”, and “jti”.

- **ref**: The name of a flow variable. If present, the policy will use the value of this variable as the claim. If both a ref attribute and an explicit claim value are specified, the explicit value is the default, and is used if the referenced flow variable is unresolved.

- **type**: One of: string (default), number, boolean, or map

- **array**: Set to true to indicate if the value is an array of types. Default: false.

Only name attribute is mandatory.
### Elements and Attributes

<table>
<thead>
<tr>
<th>Description</th>
<th>Elements and Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify additional claim in the header for the JWT.</td>
<td>AdditionalHeaders (optional)</td>
</tr>
<tr>
<td></td>
<td>&lt;AdditionalHeaders&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Claim name='claim1'&gt;explicit-value-of-claim-here&lt;/Claim&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Claim name='claim2' ref='var-name'/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Claim name='claim3' ref='var-name' type='boolean'/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Claim name='claim4' ref='var-name' type='string' array='true'/&gt;</td>
</tr>
<tr>
<td>The &lt;Claim&gt; element takes these attributes:</td>
<td></td>
</tr>
<tr>
<td>• name: name of the claim.</td>
<td></td>
</tr>
<tr>
<td>• ref: The name of a flow variable. If present, the policy uses the value of this variable as the claim. If both a ref attribute and an explicit claim value are specified, the explicit value is the default, and is used if the referenced flow variable is unresolved.</td>
<td></td>
</tr>
<tr>
<td>• type: One of: string (default), number, boolean, or map</td>
<td></td>
</tr>
<tr>
<td>• array: Set to true to indicate if the value is an array of types. Default: false.</td>
<td></td>
</tr>
<tr>
<td>Only name attribute is mandatory.</td>
<td></td>
</tr>
<tr>
<td>Specifies the lifespan of the JWT in seconds, minutes, hours, or days.</td>
<td>ExpiresIn (Optional)</td>
</tr>
<tr>
<td>Supported value: Time units can be specified as:</td>
<td></td>
</tr>
<tr>
<td>• s = seconds (default)</td>
<td></td>
</tr>
<tr>
<td>• m = minutes</td>
<td></td>
</tr>
<tr>
<td>• h = hours</td>
<td></td>
</tr>
<tr>
<td>• d = days</td>
<td></td>
</tr>
<tr>
<td>For example, an ExpiresIn = 10d is equivalent to an ExpiresIn of 864000s or 864000s.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;ExpiresIn&gt;time-value&lt;/ExpiresIn&gt;</td>
</tr>
<tr>
<td>The JWT ID (jti) claim is a unique identifier for the JWT. Id attribute generates a JWT with the specific jti claim. When the text value and ref attribute are both empty, the policy generates a jti containing a random UUID.</td>
<td>Id (Optional)</td>
</tr>
<tr>
<td></td>
<td>&lt;Id&gt;explicit-jti&lt;/Id&gt;</td>
</tr>
<tr>
<td></td>
<td>-or-</td>
</tr>
<tr>
<td></td>
<td>&lt;Id ref='var-name'/&gt;</td>
</tr>
<tr>
<td></td>
<td>-or-</td>
</tr>
<tr>
<td></td>
<td>&lt;Id/&gt;</td>
</tr>
<tr>
<td>Elements and Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IgnoreUnresolvedVariables (Optional)</td>
<td>Set to false if you want the policy to throw an error when any referenced variable specified in the policy is unresolvable. Set to true to treat any unresolvable variable as an empty string.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;IgnoreUnresolvedVariables&gt;true&lt;/IgnoreUnresolvedVariables&gt;</code></td>
</tr>
<tr>
<td>Issuer (Optional)</td>
<td>The policy generates a JWT containing a claim with name iss, with a value set to the specified value. A claim that identifies the issuer of the JWT.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Issuer ref='variable-name-here'/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Issuer&gt;issuer-string-here&lt;/Issuer&gt;</code></td>
</tr>
<tr>
<td>NotBefore (Optional)</td>
<td>Specifies an absolute time value for the token's expiration. The token is not valid until the specified time.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;NotBefore&gt;2019-06-18T11:00:11-07:00&lt;/NotBefore&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Supported values:</td>
</tr>
<tr>
<td>Name</td>
<td>Format</td>
</tr>
<tr>
<td>sortable</td>
<td><code>yyyy-MM-dd'T'HH:mm:ss.SSSZ</code></td>
</tr>
<tr>
<td>RFC 1123</td>
<td><code>EEE, dd MMM yyyy HH:mm:ss zzz</code></td>
</tr>
<tr>
<td>RFC 850</td>
<td><code>EEEE, dd-MMM-yy HH:mm:ss zzz</code></td>
</tr>
<tr>
<td>ANCI-C</td>
<td><code>EEE MMM d HH:mm:ss yyyy</code></td>
</tr>
<tr>
<td>OutputVariable (Optional)</td>
<td>JWT generated by this policy is placed in the variable specified in this attribute. By default it is placed into the flow variable message.content.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;OutputVariable&gt;jwt-var&lt;/OutputVariable&gt;</code></td>
</tr>
<tr>
<td><code>&lt;PrivateKey/Id&gt;</code> (Optional)</td>
<td>Specifies the key ID (kid) to include in the JWT header.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;PrivateKey&gt;</code>&lt;Id ref=&quot;flow-variable-name-here&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td><code>&lt;PrivateKey&gt;</code>&lt;Id&gt;your-id-value-here&lt;/Id&gt;`</td>
</tr>
</tbody>
</table>
### Elements and Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;PrivateKey/Password&gt;</code> (Optional)</td>
<td>Password to decrypt the private key, if necessary. Use the ref attribute to pass the key in a flow variable. Use this element only when the algorithm is an RSA variant.</td>
</tr>
<tr>
<td><code>&lt;PrivateKey&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;Password ref=&quot;private.privatekey-password&quot;/&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;PrivateKey/Value&gt;</code> (Optional)</td>
<td>Specifies a PEM-encoded private key used to sign the JWT. Use the ref attribute to pass the key in a flow variable. Use this only with the GenerateJWT policy, when the algorithm is an RSA variant.</td>
</tr>
<tr>
<td><code>&lt;PrivateKey&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;Value ref=&quot;private.variable-name-here&quot;/&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;SecretKey/Id&gt;</code> (Optional)</td>
<td>Specifies the key ID (kid) to include in the JWT header of a JWT signed with an HMAC algorithm.</td>
</tr>
<tr>
<td><code>&lt;SecretKey&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;Id ref=&quot;flow-variable-name-here&quot;/&gt;</code></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td><code>&lt;SecretKey&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;Id&gt;your-id-value-here&lt;/Id&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;SecretKey/Value&gt;</code> (Optional)</td>
<td>Provides the secret key used to verify or sign tokens with an HMAC algorithm. Use only when the algorithm is one of HS256, HS384, HS512. Use the ref attribute to pass the key in a flow variable.</td>
</tr>
<tr>
<td><code>&lt;SecretKey&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;Value ref=&quot;private.your-variable-name&quot;/&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;Subject&gt;</code> (Optional)</td>
<td>The policy generates a JWT containing a sub claim, set to the specified value. This claim identifies or makes a statement about the subject of the JWT.</td>
</tr>
<tr>
<td><code>&lt;Subject&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>subject-string-here&lt;/Subject&gt;</code></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Subject ref=&quot;flow_variable&quot; /&gt;</code></td>
<td></td>
</tr>
<tr>
<td>- For example -</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Subject ref=&quot;sap.developer.email&quot;/&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

The following flow variables are available after the policy is executed:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header.algorithm</td>
<td>Signing algorithm used on JWT.</td>
</tr>
<tr>
<td>claim.subject</td>
<td>JWT subject claim.</td>
</tr>
<tr>
<td>claim.issuer</td>
<td>JWT issuer claim.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>claim.audience</td>
<td>JWT audience claim. This value may be a string, or an array of strings.</td>
</tr>
<tr>
<td>claim.expiry</td>
<td>Expiration date or time, expressed in seconds since epoch.</td>
</tr>
<tr>
<td>expiry_formatted</td>
<td>Expiration date or time, formatted as a human readable string. Example:</td>
</tr>
<tr>
<td></td>
<td>2019-18-28T21:30:45.000+0000</td>
</tr>
<tr>
<td>seconds_remaining</td>
<td>Number of seconds before the token expires. If the token is expired, this</td>
</tr>
<tr>
<td></td>
<td>number will be negative.</td>
</tr>
<tr>
<td>time_remaining_formatted</td>
<td>Time remaining before the token expires, formatted as a human-readable string.</td>
</tr>
<tr>
<td></td>
<td>Example: 00:59:59.926</td>
</tr>
<tr>
<td>is_expired</td>
<td>true or false</td>
</tr>
<tr>
<td>claim.issuedat</td>
<td>Token issued date, expressed in seconds since epoch.</td>
</tr>
<tr>
<td>claim.notbefore</td>
<td>If the JWT includes a notBefore claim, this variable will contain the</td>
</tr>
<tr>
<td></td>
<td>value. This is expressed in seconds since epoch.</td>
</tr>
<tr>
<td>valid</td>
<td>In the case of VerifyJWT, this variable will be true when the signature is</td>
</tr>
<tr>
<td></td>
<td>verified, and the current time is before the token expiry, and after the</td>
</tr>
<tr>
<td></td>
<td>token notBefore value, if they are present. Otherwise false.</td>
</tr>
<tr>
<td></td>
<td>In the case of DecodeJWT, this variable is not set.</td>
</tr>
<tr>
<td>claim.name</td>
<td>The value of the named claim (standard or additional) in the payload. One</td>
</tr>
<tr>
<td></td>
<td>of these will be set for every claim in the payload.</td>
</tr>
<tr>
<td>header.name</td>
<td>The value of the named header (standard or additional). One of these will be</td>
</tr>
<tr>
<td></td>
<td>set for every additional header in the header portion of the JWT.</td>
</tr>
<tr>
<td>header.kid</td>
<td>The Key ID, if added when the JWT was generated.</td>
</tr>
<tr>
<td>header.type</td>
<td>Will be set to JWT.</td>
</tr>
<tr>
<td>payload-claim-names</td>
<td>An array of claims supported by the JWT.</td>
</tr>
<tr>
<td>payload-json</td>
<td>Payload in JSON format.</td>
</tr>
<tr>
<td>header-json</td>
<td>Header in JSON format.</td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>steps.jwt.AlgorithmInTokenNotPresentInConfiguration</td>
<td>Occurs when the verification policy has multiple algorithms.</td>
</tr>
<tr>
<td>steps.jwt.AlgorithmMismatch</td>
<td>The algorithm specified in the Generate policy didn’t match the one</td>
</tr>
<tr>
<td></td>
<td>expected in the Verify policy. The algorithms specified must match.</td>
</tr>
<tr>
<td>steps.jwt.FailedToDecode</td>
<td>The policy was unable to decode the JWT. The JWT is possibly corrupted.</td>
</tr>
<tr>
<td>steps.jwt.GenerationFailed</td>
<td>The policy was unable to generate the JWT.</td>
</tr>
<tr>
<td>steps.jwt.InsufficientKeyLength</td>
<td>For a key less than 32 bytes for the HS256 algorithm, less than 48</td>
</tr>
<tr>
<td></td>
<td>bytes for the HS386 algorithm, and less than 64 bytes for the HS512</td>
</tr>
<tr>
<td></td>
<td>algorithm.</td>
</tr>
<tr>
<td>Error Name</td>
<td>Cause</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>steps.jwt.InvalidClaim</td>
<td>For a missing claim or claim mismatch, or a missing header or header mismatch.</td>
</tr>
<tr>
<td>steps.jwt.InvalidCurve</td>
<td>The curve specified by the key is not valid for the Elliptic Curve algorithm.</td>
</tr>
<tr>
<td>steps.jwt.InvalidJsonFormat</td>
<td>Invalid JSON found in the header or payload.</td>
</tr>
<tr>
<td>steps.jwt.InvalidToken</td>
<td>This error occurs when the JWT signature verification fails.</td>
</tr>
<tr>
<td>steps.jwt.JwtAudienceMismatch</td>
<td>The audience claim failed on token verification.</td>
</tr>
<tr>
<td>steps.jwt.JwtIssuerMismatch</td>
<td>The issuer claim failed on token verification.</td>
</tr>
<tr>
<td>steps.jwt.JwtSubjectMismatch</td>
<td>The subject claim failed on token verification.</td>
</tr>
<tr>
<td>steps.jwt.KeyIdMissing</td>
<td>The Verify policy uses a JWKS as a source for public keys, but the signed JWT does not include a kid property in the header</td>
</tr>
<tr>
<td>steps.jwt.KeyParsingFailed</td>
<td>The public key could not be parsed from the given key information.</td>
</tr>
<tr>
<td>steps.jwt.NoAlgorithmFoundInHeader</td>
<td>Occurs when the JWT contains no algorithm header.</td>
</tr>
<tr>
<td>steps.jwt.NoMatchingPublicKey</td>
<td>The Verify policy uses a JWKS as a source for public keys, but the kid in the signed JWT is not listed in the JWKS.</td>
</tr>
<tr>
<td>steps.jwt.SignKeyingFailed</td>
<td>In GenerateJWT, for a key less than the minimum size for the HS384 or HS512 algorithms</td>
</tr>
<tr>
<td>steps.jwt.TokenExpired</td>
<td>The policy attempts to verify an expired token.</td>
</tr>
<tr>
<td>steps.jwt.TokenNotYetValid</td>
<td>The token isn’t yet valid.</td>
</tr>
<tr>
<td>steps.jwt.UnknownException</td>
<td>An unknown exception occurred.</td>
</tr>
<tr>
<td>steps.jwt.WrongKeyType</td>
<td>Wrong type of key specified. For example, if you specify an RSA key for an Elliptic Curve algorithm, or a curve key for an RSA algorithm.</td>
</tr>
</tbody>
</table>

The following fault variables are set when the policy triggers an error at runtime:

Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault.name=&quot;fault_name&quot;</td>
<td>fault_name is the name of the fault, as listed in the Runtime errors table above. The fault name is the last part of the fault code.</td>
<td>fault.name Matches “TokenExpired”</td>
</tr>
<tr>
<td>jwt.policy_name.failed</td>
<td>policy_name is the user-specified name of the policy that threw the fault.</td>
<td>jwt.JWT-Policy.failed = true</td>
</tr>
</tbody>
</table>

Following is an example of a fault rule:

```xml
<FaultRules>
  <FaultRule name="JWT Policy Errors">
    <Step>
      <Name>JavaScript-1</Name>
    </Step>
  </FaultRule>
</FaultRules>
```
<Condition>(fault.name Matches "TokenExpired")</Condition>
</Step>
<Condition>jwt.JWT-1.failed=true</Condition>
</FaultRule>
</FaultRules>

**Related Information**

JSON Web Tokens [page 180]
Verify JWT [page 191]
Decode JWT [page 199]

### 1.5.1.3.4.1.13.2 Verify JWT

This policy describes about Verify JSON Web Token (JWT) Policy.

This policy verifies a signed JWT, with a configurable set of claims. When this policy executes, API Management verifies the signature of a JWT, and verifies that the JWT is valid according to the expiry and not-before times if they’re present. The policy can optionally also verify the values of specific claims on the JWT, such as the subject, the issuer, the audience, or the value of additional claims. If the JWT is verified and valid, then all of the claims contained within the JWT are extracted into context variables for use by subsequent policies or conditions, and the request is allowed to proceed. If the JWT signature can’t be verified or if the JWT is invalid because of one of the timestamps, all processing stops and an error is returned in the response.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request ➔</td>
<td>Response</td>
</tr>
<tr>
<td>Preflow</td>
<td>Preflow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

**Verify a JWT signed with the HS256 algorithm**

The sample provided below verifies a JWT signed with the HS256 encryption algorithm. HMAC using a SHA-256 checksum. HS256 relies on a shared secret for both signing and verifying the signature.

```xml
<VerifyJWT async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <Algorithm>HS256</Algorithm>
  <Source>request.formparam.jwt</Source>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <SecretKey>
    <Value ref="private.secretkey"/>
  </SecretKey>
</VerifyJWT>
```
Verify a JWT signed with the RS256 algorithm

This example policy verifies a JWT that was signed using the RS256 algorithm. For signing, a private key must be provided, and to verify, you need to provide the corresponding public key.

**Code Syntax**

```xml
<VerifyJWT async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <Algorithm>RS256</Algorithm>
  <Source>request.formparam.jwt</Source>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <PublicKey>
    <Value ref="public.publickey"/>
  </PublicKey>
  <Subject>subject-subject</Subject>
  <Issuer>urn://apim-JWT-policy-test</Issuer>
  <Audience>audience1,audience2</Audience>
  <AdditionalClaims>
    <Claim name="additional-claim-name" type="string">additional-claim-value-goes-here</Claim>
  </AdditionalClaims>
</VerifyJWT>
```

The resulting JWT will have this header and payload and is valid, if the signature can be verified with the provided public key.

**Sample Code**

```json
{
  "typ": "JWT",
  "alg": "RS256"
}

{
  "sub": "subject-subject",
  "iss": "urn://apim-edge-JWT-policy-test",
  "aud": "audience1,audience2",
  "additional-claim-name": "additional-claim-value-goes-here"
}
```

However, a JWT with the same header but different payload as shown below is invalid, even if the signature is verified. The "sub" claim included in the JWT does not match the required value of the "Subject" element as specified in the policy configuration.

**Sample Code**

```json
{
}
```
Following table lists the elements and attributes that you can configure on this policy

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm</strong></td>
<td>Specifies the encryption algorithm to sign the token. HS256 employs a shared secret. RS256 employs a public/secret key pair. Supported value: HS256, HS384, HS512, RS256, RS384, RS512</td>
</tr>
<tr>
<td><strong>Audience (optional)</strong></td>
<td>The policy verifies that the audience claim in the JWT matches the value specified in the configuration. If there is no match, the policy throws an error.</td>
</tr>
<tr>
<td><strong>AdditionalClaims (optional)</strong></td>
<td>Validates additional claims in the payload of the JWT. An additional claim can be a string, a number, a boolean, a map, or an array. A map is simply a set of name/value pairs.</td>
</tr>
</tbody>
</table>

```xml
<AdditionalClaims>
  <Claim name='claim1'>explicit-value-of-claim-here</Claim>
  <Claim name='claim2' ref='var-name'/>
  <Claim name='claim3' ref='var-name' type='boolean'/>
</AdditionalClaims>
```

The `<Claim>` element takes these attributes:

- **name**: name of the claim.
- **ref**: The name of a flow variable. If present, the policy will use the value of this variable as the claim. If both a ref attribute and an explicit claim value are specified, the explicit value is the default, and is used if the referenced flow variable is unresolved.
- **type**: One of: string (default), number, boolean, or map
- **array**: Set to true to indicate if the value is an array of types. Default: false.

Note:

Don’t use any of the registered claim names in this element. They include: "iss", "sub", "aud", "iat", "exp", "nbf", and "jti".

Only name attribute is mandatory.
<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdditionalHeaders (optional)</td>
<td>Validates additional claim in the header for the JWT.</td>
</tr>
<tr>
<td></td>
<td>&lt;AdditionalHeaders&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Claim name='claim1'&gt;explicit-value-of-claim-here&lt;/Claim&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Claim name='claim2' ref='var-name'/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Claim name='claim3' ref='var-name' type='boolean'/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Claim name='claim4' ref='var-name' type='string' array='true'/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/AdditionalHeaders&gt;</td>
</tr>
</tbody>
</table>

The `<Claim>` element takes these attributes:

- **name**: name of the claim.
- **ref**: The name of a flow variable. If present, the policy uses the value of this variable as the claim. If both a ref attribute and an explicit claim value are specified, the explicit value is the default, and is used if the referenced flow variable is unresolved.
- **type**: One of: string (default), number, boolean, or map
- **array**: Set to true to indicate if the value is an array of types. Default: false.

Only name attribute is mandatory.

<table>
<thead>
<tr>
<th>Id (Optional)</th>
<th>The JWT ID (jti) claim is a unique identifier for the JWT. Id attribute verifies if the JWT contains the specific jti claim. When the text value and ref attribute are both empty, the policy generates a jti containing a random UUID.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;Id&gt;explicit-jti&lt;/Id&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Id ref='varname'/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Id/&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IgnoreUnresolvedVariables (Optional)</th>
<th>Set to false if you want the policy to throw an error when any referenced variable specified in the policy is unresolvable. Set to true to treat any unresolvable variable as an empty string</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;IgnoreUnresolvedVariables&gt;true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Issuer (Optional)</th>
<th>The policy verifies that the issuer in the JWT matches the string specified in the configuration element.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;Issuer ref='variable-name-here'/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Issuer&gt;issuer-string-here&lt;/Issuer&gt;</td>
</tr>
</tbody>
</table>
### Elements and Attributes

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;PublicKey/JWKS&gt;</code> (JWKS or Value element is required to verify a JWT signed with RSA algorithm)</td>
<td>Specifies a value in the JSON web key set (JWKS) format containing a set of public keys. If the JWT contains a key ID in the set of JWKS, then the policy uses the correct public key to verify the JWT.</td>
</tr>
<tr>
<td><code>&lt;PublicKey/JWKS&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;PublicKey&gt;</code></td>
<td>Use this element only when the algorithm is an RSA variant. This element specifies the public key to verify the signature. Specify the PEM-encoded key directly or use the ref attribute to pass the key in a flow variable.</td>
</tr>
<tr>
<td><code>&lt;Source&gt;</code> (Optional)</td>
<td>If present, specifies the flow variable in which the policy expects to find the JWT to verify.</td>
</tr>
</tbody>
</table>

**Restriction**

JWT validation fails RSA keys smaller than 2048 bits. Ensure that your keys are 2048 bits or larger.

```xml
<PublicKey>
  <Source>jwt-variable</Source>
</PublicKey>
```
### Elements and Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Subject&gt;</code> (Optional)</td>
<td>The policy verifies that the subject in the JWT matches the string specified in the policy configuration.</td>
</tr>
<tr>
<td><code>&lt;SecretKey/Value&gt;</code> (Optional)</td>
<td>Provides the secret key used to verify or sign tokens with an HMAC algorithm. Use only when the algorithm is one of HS256, HS384, HS512. Use the ref attribute to pass the key in a flow variable.</td>
</tr>
<tr>
<td><code>&lt;TimeAllowance&gt;</code> (Optional)</td>
<td>The &quot;grace period&quot; for times. For example, if the time allowance is configured to be 60s, then an expired JWT would be treated as still valid, for 60s after the asserted expiry. The not-before-time will be evaluated similarly. Default value is 0s.</td>
</tr>
</tbody>
</table>

The following flow variables are available after the policy is executed:

#### Flow Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header.algorithm</td>
<td>Signing algorithm used on JWT.</td>
</tr>
<tr>
<td>claim.subject</td>
<td>JWT subject claim.</td>
</tr>
<tr>
<td>claim.issuer</td>
<td>JWT issuer claim.</td>
</tr>
<tr>
<td>claim.audience</td>
<td>JWT audience claim. This value may be a string, or an array of strings.</td>
</tr>
<tr>
<td>claim.expiry</td>
<td>Expiration date or time, expressed in seconds.</td>
</tr>
<tr>
<td>expiryFormatted</td>
<td>Expiration date or time, formatted as a human readable string. Example: 2019-18-28T21:30:45.000+0000</td>
</tr>
<tr>
<td>seconds_remaining</td>
<td>Number of seconds before the token expires. If the token is expired, this number will be negative.</td>
</tr>
<tr>
<td>time_remainingFormatted</td>
<td>Time remaining before the token expires, formatted as a human-readable string. Example: 00:59:59.926</td>
</tr>
<tr>
<td>is_expired</td>
<td>true or false</td>
</tr>
<tr>
<td>claim.issuedat</td>
<td>Token issued Date, expressed in seconds since epoch.</td>
</tr>
<tr>
<td>claim.notbefore</td>
<td>If the JWT includes a nbf claim, this variable will contain the value. This is expressed in seconds since epoch.</td>
</tr>
<tr>
<td>valid</td>
<td>In the case of VerifyJWT, this variable will be true when the signature is verified, and the current time is before the token expiry, and after the token notBefore value, if they are present. Otherwise false. In the case of DecodeJWT, this variable is not set.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>claim.name</td>
<td>The value of the named claim (standard or additional) in the payload. One of these will be set for every claim in the payload.</td>
</tr>
<tr>
<td>header.name</td>
<td>The value of the named header (standard or additional). One of these will be set for every additional header in the header portion of the JWT.</td>
</tr>
<tr>
<td>header.kid</td>
<td>The Key ID, if added when the JWT was generated.</td>
</tr>
<tr>
<td>header.type</td>
<td>Will be set to JWT.</td>
</tr>
<tr>
<td>payload-claim-names</td>
<td>An array of claims supported by the JWT.</td>
</tr>
<tr>
<td>payload-json</td>
<td>Payload in JSON format.</td>
</tr>
<tr>
<td>header-json</td>
<td>Header in JSON format.</td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Cause</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>steps.jwt.AlgorithmInTokenNotPresentInConfiguration</td>
<td>Occurs when the verification policy has multiple algorithms.</td>
</tr>
<tr>
<td>steps.jwt.AlgorithmMismatch</td>
<td>The algorithm specified in the Generate policy did not match the one expected in the Verify policy. The algorithms specified must match</td>
</tr>
<tr>
<td>steps.jwt.FailedToDecode</td>
<td>The policy was unable to decode the JWT. The JWT is possibly corrupted.</td>
</tr>
<tr>
<td>steps.jwt.GenerationFailed</td>
<td>The policy was unable to generate the JWT.</td>
</tr>
<tr>
<td>steps.jwt.InsufficientKeyLength</td>
<td>For a key less than 32 bytes for the HS256 algorithm, less than 48 bytes for the HS386 algorithm, and less than 64 bytes for the HS512 algorithm.</td>
</tr>
<tr>
<td>steps.jwt.InvalidClaim</td>
<td>For a missing claim or claim mismatch, or a missing header or header mismatch.</td>
</tr>
<tr>
<td>steps.jwt.InvalidCurve</td>
<td>The curve specified by the key is not valid for the Elliptic Curve algorithm.</td>
</tr>
<tr>
<td>steps.jwt.InvalidJsonFormat</td>
<td>Invalid JSON found in the header or payload.</td>
</tr>
<tr>
<td>steps.jwt.JwtAuthenticationMismatch</td>
<td>This error occurs when the JWT signature verification fails.</td>
</tr>
<tr>
<td>steps.jwt.JwtIssuerMismatch</td>
<td>The issuer claim failed on token verification.</td>
</tr>
<tr>
<td>steps.jwt.JwtSubjectMismatch</td>
<td>The subject claim failed on token verification.</td>
</tr>
<tr>
<td>steps.jwt.KeyIdMissing</td>
<td>The Verify policy uses a JWKS as a source for public keys, but the signed JWT does not include a kid property in the header.</td>
</tr>
<tr>
<td>steps.jwt.KeyParsingFailed</td>
<td>The public key could not be parsed from the given key information.</td>
</tr>
<tr>
<td>steps.jwt.NoAlgorithmFoundInHeader</td>
<td>Occurs when the JWT contains no algorithm header.</td>
</tr>
<tr>
<td>steps.jwt.NoMatchingPublicKey</td>
<td>The Verify policy uses a JWKS as a source for public keys, but the kid in the signed JWT is not listed in the JWKS.</td>
</tr>
<tr>
<td>Error Name</td>
<td>Cause</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>steps.jwt.SigningFailed</td>
<td>In GenerateJWT, for a key less than the minimum size for the</td>
</tr>
<tr>
<td></td>
<td>HS384 or HS512 algorithms</td>
</tr>
<tr>
<td>steps.jwt.TokenExpired</td>
<td>The policy attempts to verify an expired token.</td>
</tr>
<tr>
<td>steps.jwt.TokenNotYetValid</td>
<td>The token is not yet valid.</td>
</tr>
<tr>
<td>steps.jwt.UnknownException</td>
<td>An unknown exception occurred.</td>
</tr>
<tr>
<td>steps.jwt.WrongKeyType</td>
<td>Wrong type of key specified. For example, if you specify an RSA key</td>
</tr>
<tr>
<td></td>
<td>for an Elliptic Curve algorithm, or a curve key for an RSA algorithm.</td>
</tr>
</tbody>
</table>

The following fault variables are set when the policy triggers an error at runtime:

**Fault Variables**

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault.name=&quot;fault_name&quot;</td>
<td>fault_name is the name of the fault, as listed in the Runtime errors table above. The fault name is the last part of the fault code.</td>
<td>fault.name Matches &quot;TokenExpired&quot;</td>
</tr>
<tr>
<td>jwt.policy_name.failed</td>
<td>policy_name is the user-specified name of the policy that threw the fault.</td>
<td>jwt.JWT-Policy.failed = true</td>
</tr>
</tbody>
</table>

Following is an example of a fault rule:

```xml
<FaultRules>
  <FaultRule name="JWT Policy Errors">
    <Step>
      <Name>JavaScript-1</Name>
      <Condition>(fault.name Matches "TokenExpired")</Condition>
    </Step>
    <Condition>jwt.JWT-1.failed=true</Condition>
  </FaultRule>
</FaultRules>
```

**Related Information**

- JSON Web Tokens [page 180]
- Generate JWT [page 182]
- Decode JWT [page 199]
1.5.1.3.4.1.13.3 Decode JWT

This policy describes about Decode JSON Web Token (JWT) Policy.

This policy verifies a signed JWT, with a configurable set of claims. When this policy executes, API Management verifies the signature of a JWT, and verifies that the JWT is valid according to the expiry and not-before times if they are present. The policy can optionally also verify the values of specific claims on the JWT, such as the subject, the issuer, the audience, or the value of additional claims. If the JWT is verified and valid, then all of the claims contained within the JWT are extracted into context variables for use by subsequent policies or conditions, and the request is allowed to proceed. If the JWT signature cannot be verified or if the JWT is invalid because of one of the timestamps, all processing stops and an error is returned in the response.

You can attach this policy in the following locations:

```
<DecodeJWT async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgt">
  <Source>var.jwt</Source>
</DecodeJWT>
```

The following table lists the elements and attributes that you can configure on this policy:

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Source&gt; (Optional)</td>
<td>If present, specifies the flow variable in which the policy expects to find the JWT to decode.</td>
</tr>
<tr>
<td></td>
<td>&lt;Source&gt;jwt-variable&lt;/Source&gt;</td>
</tr>
</tbody>
</table>

The following flow variables are available after the policy is executed:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header.algorithm</td>
<td>Signing algorithm used on JWT.</td>
</tr>
<tr>
<td>claim.subject</td>
<td>JWT subject claim.</td>
</tr>
<tr>
<td>claim.issuer</td>
<td>JWT issuer claim.</td>
</tr>
<tr>
<td>claim.audience</td>
<td>JWT audience claim. This value may be a string, or an array of strings.</td>
</tr>
</tbody>
</table>
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>claim.expiry</td>
<td>Expiration date or time, expressed in seconds.</td>
</tr>
<tr>
<td>expiry_formatted</td>
<td>Expiration date or time, formatted as a human readable string. Example: 2019-18-28T21:30:45.000+0000</td>
</tr>
<tr>
<td>seconds_remaining</td>
<td>Number of seconds before the token expires. If the token is expired, this number will be negative.</td>
</tr>
<tr>
<td>time_remaining_formatted</td>
<td>Time remaining before the token expires, formatted as a human-readable string. Example: 00:59:59.926</td>
</tr>
<tr>
<td>is_expired</td>
<td>true or false</td>
</tr>
<tr>
<td>claim.issuedat</td>
<td>Token issued Date, expressed in seconds since epoch.</td>
</tr>
<tr>
<td>claim.notbefore</td>
<td>If the JWT includes a nbf claim, this variable will contain the value. This is expressed in seconds since epoch.</td>
</tr>
<tr>
<td>valid</td>
<td>In the case of VerifyJWT, this variable will be true when the signature is verified, and the current time is before the token expiry, and after the token notBefore value, if they are present. Otherwise false. In the case of DecodeJWT, this variable is not set.</td>
</tr>
<tr>
<td>claim.name</td>
<td>The value of the named claim (standard or additional) in the payload. One of these will be set for every claim in the payload.</td>
</tr>
<tr>
<td>header.name</td>
<td>The value of the named header (standard or additional). One of these will be set for every additional header in the header portion of the JWT.</td>
</tr>
<tr>
<td>header.kid</td>
<td>The Key ID, if added when the JWT was generated.</td>
</tr>
<tr>
<td>header.type</td>
<td>Will be set to JWT.</td>
</tr>
<tr>
<td>payload-claim-names</td>
<td>An array of claims supported by the JWT.</td>
</tr>
<tr>
<td>payload-json</td>
<td>Payload in JSON format.</td>
</tr>
<tr>
<td>header-json</td>
<td>Header in JSON format.</td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>steps.jwt.FailedToDecode</td>
<td>Occurs when the policy is unable to decode the JWT. The JWT may be malformed, invalid or otherwise not decodable.</td>
</tr>
<tr>
<td>steps.jwt.InvalidToken</td>
<td>Occurs when the flow variable specified in the &lt;Source&gt; element of the policy is out of scope or can't be resolved.</td>
</tr>
</tbody>
</table>

The following fault variables are set when the policy triggers an error at runtime:
Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault.name=&quot;fault_name&quot;</td>
<td>fault_name is the name of the fault, as listed in the Runtime errors table above. The fault name is the last part of the fault code.</td>
<td>fault.name Matches &quot;TokenExpired&quot;</td>
</tr>
<tr>
<td>jwt.policy_name.failed</td>
<td>policy_name is the user-specified name of the policy that threw the fault.</td>
<td>jwt.JWT-Policy.failed = true</td>
</tr>
</tbody>
</table>

Following is an example of a fault rule:

```
<FaultRules>
  <FaultRule name="JWT Policy Errors">
    <Step>
      <Name>JavaScript-1</Name>
      <Condition>(fault.name Matches "TokenExpired")</Condition>
    </Step>
    <Condition>jwt.JWT-1.failed=true</Condition>
  </FaultRule>
</FaultRules>
```

Related Information

- JSON Web Tokens [page 180]
- Generate JWT [page 182]
- Verify JWT [page 191]

1.5.1.3.4.1.14 Key Value Map Operations

The Key Value Map Operations policy allows you to create a key value map and perform update, read, and delete operations on the map.

Key Value Map Operations are typically used to store or retrieve long-lived information that needs to be reused over multiple request or response transactions.

KeyValuel refers to any arbitrary data in the format key=value, for example localhost=127.0.0.1, zip_code=94110, or first_name=Philip.

In the first example, localhost is a key, and 127.0.0.1 is a value.

Each key/value pair is stored in a map as an entry. A key/value map can store many entries. For example, say you need to store a list of IP addresses associated with various backend environments.
You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>Preflow</td>
<td>Preflow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<KeyValueMapOperations async="true" continueOnError="false"
    enabled="true" mapIdentifier="urlMapper" xmlns="http://www.sap.com/apimgmt">
    <InitialEntries>
    <Entry>
        <Key>
            <Parameter>key1</Parameter>
        </Key>
        <Value>val1</Value>
    </Entry>
    <Entry>
        <Key>
            <Parameter>var_name</Parameter>
        </Key>
        <Value>val1</Value>
        <Value>val2</Value>
    </Entry>
    </InitialEntries>
    <Put override="false">
        <Key>
            <Parameter>key1</Parameter>
        </Key>
        <Value ref="var_name"/>
    </Put>
    <Get assignTo="sapapim.empnumber" index="1">
        <Key>
            <Parameter ref="sapapim.empEmail"/>
        </Key>
    </Get>
    <Delete>
        <Key>
            <Parameter>key1</Parameter>
        </Key>
    </Delete>
    </KeyValueMapOperations>
```

### `<mapIdentifier>` element

<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapIdentifier</td>
<td>N/A</td>
<td>N/A</td>
<td>Specifies an identifier to be used when accessing a map created by this policy.</td>
</tr>
</tbody>
</table>
If you exclude this attribute, a KVM named `kvmap` is used.

### <Scope> element

<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope (Optional)</td>
<td>environment</td>
<td>String</td>
<td>Defines the boundary of accessibility for Key Value Maps. The default value is environment. That is, by default, maps entries are shared by all API proxies running in an environment. The valid values are apiproxy, environment, organization and policy. If you set the scope to apiproxy, then the entries in the key value map are accessible only by the API Proxy that writes the values to the map.</td>
</tr>
</tbody>
</table>

**Note**

When accessing a map or map entry, you must specify the same scope value you used when the map was created. For example, if the map was created with a scope of apiproxy, you must use the apiproxy scope when retrieving its values, updating changes, or deleting entries.

### <Entry> element

<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry (Optional)</td>
<td>N/A</td>
<td>N/A</td>
<td>Seed values for key value maps, which are populated in the key value map when it’s initialized.</td>
</tr>
</tbody>
</table>
### `<InitialEntries>` element

<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialEntries</td>
<td>N/A</td>
<td>N/A</td>
<td>Seed values for key value maps, which are populated in the key value map when it’s initialized. Make sure to specify the name of the KVM with the <code>mapIdentifier</code> attribute on the parent element. When using this element, when you save the policy on a deployed version of the proxy, or deploy the API proxy bundle containing the policy with this element, the key(s) are automatically created in the KVM (as unencrypted). If the values in...</td>
</tr>
</tbody>
</table>
the policy are different than the values in the KVM, the values in the KVM are overwritten when the proxy is deployed. Any new keys/values are added to the existing KVM alongside the existing keys/values.

**Sample Code**

```xml
<InitialEntries>
  <Entry>
    <Key>
      <Parameter>key1</Parameter>
    </Key>
    <Value>val1</Value>
  </Entry>
  <Entry>
    <Key>
      <Parameter>key2_variable</Parameter>
    </Key>
    <Value>val2</Value>
    <Value>val3</Value>
  </Entry>
</InitialEntries>
```

**Note**

Keys and values populated by this element must be literals.

**Sample Code**

```xml
(Parameter ref="request.queryparam.key")
```
The example shown above isn’t supported within this element.

### <key> element

<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key (Optional)</td>
<td>N/A</td>
<td>N/A</td>
<td>Specifies the key in a key/value map entry. A key can be composite, which means that more than one parameter can be appended to create the key. For example, userID and role might be combined to create a key.</td>
</tr>
</tbody>
</table>

**Sample Code**

```xml
<Key>
    <Parameter>key 1</Parameter>
</Key>
```

See the `<parameter>` element for information about how to set the key name.

### <parameter> element

<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| parameter (Required) | N/A     | String | Specifies the key in a key/value pair. This element specifies the name when creating, putting, retrieving, or deleting the key/value pair.  
  - A literal string |

**Sample Code**

```xml
<Key>
</Key>
```
<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>● A variable to be retrieved at runtime</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● A combination of variable references and literal strings</td>
</tr>
</tbody>
</table>

When the Key element includes multiple Parameter elements, the effective key string is the concatenation of the values of each parameter, joined with a double underscore. For example, in the above example, if the api_proxy.name variable has the value “def23”, then the effective key will be

```xml
<Key>
  <Parameter ref="var_name"/>
  <Parameter ref="api_proxy.name"/>
  <Parameter size/>
</Key>
```
Whether you’re retrieving, updating, or deleting a key/value entry, the key name must match the name of the key in the key value map.

ref attribute of the <parameter> element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>N/A</td>
<td>N/A</td>
<td>Specifies the name of a variable whose value contains the exact name of the key you want to create, retrieve, or delete.</td>
</tr>
</tbody>
</table>

<value> element

<table>
<thead>
<tr>
<th>Element</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value (Required)</td>
<td>N/A</td>
<td>String</td>
<td>Specifies the value of a key. You can specify the value as either a literal string or, using the ref attribute, as a variable to be retrieved at runtime.</td>
</tr>
</tbody>
</table>

Sample Code

<!-- Specify a literal value -->
<Value>literal string</Value>

<!-- Specify the name of variable value to be populated at runtime. -->
<Value>ref</Value>
You can include multiple `<value>` elements to specify a multi-part value. Values are combined at run time.

In the following example, two keys 'key1' with values 'val1' and 'val2' and 'key2' with values 'val3' and 'val4' are added to the KVM:

```xml
<InitialEntries>
    <Entry>
        <Key>
            <Parameter>key1</Parameter>
        </Key>
        <Value>val1</Value>
        <Value>val2</Value>
    </Entry>
    <Entry>
        <Key>
            <Parameter>key2</Parameter>
        </Key>
        <Value>val3</Value>
        <Value>val4</Value>
    </Entry>
</InitialEntries>
```

ref attribute of the `<value>` element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>N/A</td>
<td>N/A</td>
<td>Specifies the name of a variable whose value</td>
</tr>
<tr>
<td>Attribute</td>
<td>Default</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>contains the key value(s) you want to set.</td>
</tr>
</tbody>
</table>

### <Get> element

**Element**

Get (Required if <Put> or <Delete> are not present)

**Description**

Retrieves the value for the key specified.

- At least one of <Get>, <Put>, or <Delete> must be used.
- Make sure to specify the name of the KVM with the mapIdentifier attribute on the parent element.

**Attributes**

- **assignTo**
  - The variable to which the retrieved value must be assigned.
- **index**
  - You can specify an index for a multivalued key. For example, if index=1, the value at index 1 is fetched and assigned to the variable. If not specified, all the values of that entry are assigned to the variable as java.util.List.

#### Sample Code

```xml
<Get assignTo="var5" index="1">
  <Key>
    <Parameter>key1</Parameter>
  </Key>
</Get>
```

### <Put> element

**Element**

Put (Required if <Get> or <Delete> are not present)

**Description**

Writes a key/value pair to a key value map.

**Attributes**

- **override**
  - The default value is false. If true, it overrides the value of a key.

#### Sample Code

```xml
<Put override="false">
  <Key>
    <Parameter ref="mykeyvariable"/>
  </Key>
</Put>
```
During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetVariableFailed</td>
<td>Failed to set variable (0) in KeyValueMapStepDefinition (1). When getting values in encrypted key value maps, this error occurs if you fail to prefix the assignTo variable with &quot;private&quot;. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;Get assignTo=&quot;private.encryptedVar&quot; index=&quot;1&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Key&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Parameter&gt;foo&lt;/Parameter&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Key&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Get&gt;</td>
</tr>
<tr>
<td>RemoveVariableFailed</td>
<td>Failed to remove variable (0) in KeyValueMapStepDefinition (1)</td>
</tr>
</tbody>
</table>
### 1.5.1.3.4.1.15 Lookup Cache

An OAuth access token is written to the cache using a Populate Cache policy. The OAuth token is retrieved for subsequent requests by a Lookup Cache policy.

At runtime, the LookupCache policy retrieves a value from the cache, assigning the value to the variable you specify with the AssignTo element (if no value is retrieved, the variable will not be set). It looks for the value based on a cache key created through configuration that combines the CacheKey and Scope elements. In other words, to retrieve a particular value-added to the cache by a PopulateCache policy, your LookupCache policy must have cache key-related elements configured in the same way as the PopulateCache policy.

You can retrieve cached values with the Lookup Cache policy.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request →</td>
<td>Response</td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LookupCache async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <CacheKey>
    <KeyFragment ref="request.header.userid"></KeyFragment>
  </CacheKey>
  <Scope>Exclusive</Scope>
  <AssignTo>cache-response</AssignTo>
</LookupCache>
```

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheKey</td>
<td>Configures a unique pointer to a piece of data stored in the cache.</td>
</tr>
<tr>
<td>CacheResource</td>
<td>Specifies the cache where messages should be stored. A default cache is available.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Scope</td>
<td>Enumeration used to construct a prefix for a cache key when a Prefix element is not provided in the CacheKey element. The list of supported values are: Global, Application, Proxy, Target, and Exclusive.</td>
</tr>
<tr>
<td>AssignTo</td>
<td>Specifies the variable where the cache entry is assigned after it has been retrieved from the cache.</td>
</tr>
<tr>
<td>Prefix</td>
<td>Specifies a value to use as a cache key prefix.</td>
</tr>
<tr>
<td>KeyFragment</td>
<td>Specifies a value that should be included in the cache key, creating a namespace for matching requests to cached responses.</td>
</tr>
</tbody>
</table>

The following predefined Flow variables are available after you customize the behavior of the cache you define in a Lookup Cache policy.

**Flow Variables**

<table>
<thead>
<tr>
<th>Variables</th>
<th>Type</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lookupcache.{policy-name}.cachename</td>
<td>String</td>
<td>Read-Only</td>
<td>Returns the cache name used in the policy.</td>
</tr>
<tr>
<td>lookupcache.{policy-name}.cachekey</td>
<td>String</td>
<td>Read-Only</td>
<td>Returns the key used.</td>
</tr>
<tr>
<td>lookupcache.{policy-name}.cachehit</td>
<td>Boolean</td>
<td>Read-Only</td>
<td>True if the policy found a value for the specified cache key.</td>
</tr>
<tr>
<td>lookupcache.{policy-name}.assignto</td>
<td>String</td>
<td>Read-Only</td>
<td>Returns the variable to which cache is assigned.</td>
</tr>
</tbody>
</table>

Lookup Cache policy type defines the following error codes:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Occurs when</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidCacheResourceReference</td>
<td>The cache specified in the &lt;CacheResource&gt; element does not exist.</td>
</tr>
<tr>
<td>InvalidTimeout</td>
<td>The CacheLookupTimeoutInSeconds value must be greater than zero.</td>
</tr>
<tr>
<td>CacheNotFound</td>
<td>The cache specified in the &lt;CacheResource&gt; element does not exist.</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.16 Message Logging Policy

The Message Logging policy lets you send syslog messages to third-party log management services, such as Splunk, Sumo Logic, and Loggly.

If you want to send syslog to one of those services, follow the service documentation of the concerned service to obtain the host, port, and protocol, then set the <Syslog> element on this policy accordingly.
You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request →</td>
<td>Response</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>PostClientFlow</td>
<td></td>
<td>Preflow</td>
<td>Flow</td>
<td>PostFlow</td>
<td>Preflow</td>
<td>Flow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<MessageLogging async="false" continueOnError="false" enabled="true"
xmlns='http://www.sap.com/apimgmt'>
  <Syslog>
    <Message>Message.id = {request.header.id}</Message>
    <!-- Host must be valid DNS/IP -->
    <!-- <Host>127.0.0.1</Host> -->
    <Host></Host>
    <!-- This is default port value -->
    <Port>514</Port>
    <Protocol>TCP</Protocol>
  </Syslog>
</MessageLogging>
```

**i Note**
When using Loggly, `<FormatMessage>true</FormatMessage>` is required in the policy as a child of the `<Syslog>` element.

### Elements and Attributes

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build the message to be sent to the syslog, combining text with variables to capture the information you want.</td>
</tr>
</tbody>
</table>

- **Syslog destination**
- **Message**

To send syslog to Splunk, Sumo Logic, or Loggly.

**i Note**
Response variables will not be available in PostClientFlow following an Error Flow. Use message variables to log response information for both error and success situations.
<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>The hostname or IP address of the server where the syslog should be sent. If the element is not specified, the default is localhost.</td>
</tr>
<tr>
<td>Port</td>
<td>Port where the syslog is running. If you don’t include this element, the default is 514.</td>
</tr>
<tr>
<td>Protocol</td>
<td>TCP or UDP (default). While UDP is more performant, the TCP protocol guarantees message log delivery to the syslog server. For sending syslog messages over TLS/SSL, only TCP is supported.</td>
</tr>
</tbody>
</table>
| FormatMessage           | true or false  
Optional, but `<FormatMessage>true</FormatMessage>` is required for use with Loggly.  
This element lets you control the format of generated content prepended to the message. If set to true, the syslog message is prepended by a fixed number of characters, which lets you filter out that information from messages. |
<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLInfo</td>
<td>If you don’t include this element or leave it empty, the default value is false. Lets you log messages through SSL/TLS. Use with sub elements</td>
</tr>
</tbody>
</table>

- **Enabled**: Indicates whether TLS/SSL is enabled for the endpoint. The default value is false.
- **ClientAuthEnabled**: Outbound client authentication (2-way TLS/SSL)
- **Keystore**: A keystore containing private keys used for outbound client authentication
- **KeyAlias**: The key alias of the private key used for outbound client authentication
- **TrustStore**: A keystore containing trusted server certificates.
- **Ciphers**: Supported ciphers for outbound TLS/SSL. To restrict ciphers, add the following elements listing the supported ciphers:

  **Sample Code**

  ```xml
  <Ciphers>
  <Cipher>TLS_RSA_WITH_3DES_EDE_CBC_SHA</Cipher>
  <Cipher>TLS_RSA_WITH_DES_CBC_SHA</Cipher>
  </Ciphers>
  ```

  **Protocols**: Supported protocols for outbound TLS/SSL. To restrict protocols, add the following elements listing the supported protocols:

  **Sample Code**

  ```xml
  <Protocols>
  <Protocol>TLSv1</Protocol>
  </Protocols>
  ```
1.5.1.3.4.1.17 Quota

The Quota policy defines the number of request messages an application can submit to an API over a given period of time.

The period of time can be an hour, a day, or a month and so on. You can apply this policy on the context of request messages.

The Quota policy helps API Providers to restrict the number of calls made to an API. For example, you can restrict access to your applications by defining the number of API calls made as 20 per day or 20,000 over a period of one week.

API Management maintains a counter that keeps track of the number of calls made to the API. Once the counter reaches the Quota limit, all successive calls made to the API are rejected. API Management returns an error message to an API-call made after the Quota limit is exceeded. The Quota policy provides the capability to reset the counters automatically after the stipulated period of time, unless it is explicitly reset by using the Reset Quota policy.
1.5.1.3.4.1.17.1 Types of Quota

Quota type is an attribute of the Quota policy that you define while configuring the policy. API Management supports the following three types of Quota:

- **calendar** type of Quota. For example, `<Quota name="DemoQuota" type="calendar">`.
  In the calendar Quota, you explicitly provide a start time. The counter starts the count from the specified start date. The Quota counter is refreshed based on the Interval and TimeUnit that you set. For example, the following Quota of type calendar begins counting at 8.30 am on June 26, 2015, and will refresh once in every 20 minutes.

  ```xml
  <Quota type="calendar">
    <Identifier ref="request.header.clientId"/>
    <StartTime>2015-06-26 08:30:00</StartTime>
    <Interval>20</Interval>
    <TimeUnit>minute</TimeUnit>
    <Allow count="99"/>
    <MessageWeight ref="request.header.weight"/>
    <Distributed>true</Distributed>
    <Synchronous>true</Synchronous>
  </Quota>
  ```

  **Note**
  If the type is not mentioned, then the Quota defaults to calendar type.

- **rollingwindow**: A "rolling window" counter advances by the time interval that you specify. You do not specify a StartTime; instead, the StartTime for the counter is the time when the first message is received from the client plus the interval that you define. A counter is kept for each client ID (consumer key). Thus, the counter will reset to zero when the interval you define has passed. This enables you to configure a Quota in which an app is indefinitely allowed 1000 requests every 24 hours.

- **flexi**: Flexible Quota enforcement causes the counter to begin when the first request message is received from an app. Under flexible Quota enforcement, StartTime is dynamic; every app has its own StartTime based on the time when the first request is received. This enables you to provide Quotas that support one week, one month, or 6 months access to your API, customized for each app.
1.5.1.3.4.1.17.2 Static and Dynamic Settings

A Quota can be static or dynamic.

Static Settings

For a static quota, you provide a count, a time interval, and a time unit. In the example below, the application accepts 5000 requests per week.

**Sample Code**

```
<Quota>
  <Allow count="5000"/>
  <Interval>1</Interval>
  <TimeUnit>week</TimeUnit>
</Quota>
```

Dynamic Settings

Dynamic Quotas enable you to configure a single Quota policy that enforces different Quota settings for different applications; based on the identity of the requesting application. Dynamic Quota values are populated at runtime by resolving an application identifier to an API product. The Identifier can be a field in the request that is unique to each application. For API proxies where OAuth is enforced, you can use consumer_id as the Identifier, as demonstrated in the sample policy below:

**Sample Code**

```
<Quota>
  <Interval ref="apiproduct.developer.quota.interval"/>
  <TimeUnit ref="apiproduct.developer.quota.timeunit"/>
  <Allow countRef="apiproduct.developer.quota.limit"/>
  <Identifier ref=" consumer_id"/>
</Quota>
```

The example above uses the variable consumer_id to identify the requesting application. This works as long as the request message contains a consumer_id (associated with an OAuth-enabled request).

Related Information

Designing Quota Policy [page 220]
### 1.5.1.3.4.1.17.3 Designing Quota Policy

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>Preflow</td>
<td>Flow</td>
</tr>
<tr>
<td>Flow</td>
<td>Preflow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>Flow</td>
</tr>
<tr>
<td>Flow</td>
<td>Preflow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>Flow</td>
</tr>
<tr>
<td>Flow</td>
<td>Preflow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```
<Quota type="calendar" async="true" continueOnError="true" enabled="true">
  <Identifier ref="{ref}"></Identifier>
  <MessageWeight ref="{ref}"></MessageWeight>
  <Allow count="{count}" countRef="{countref}">
    <SyncIntervalInSeconds>{value}</SyncIntervalInSeconds>
    <SyncMessageCount>{count}</SyncMessageCount>
  </Allow>
  <AsynchronousConfiguration>
    <Interval ref="{ref}" type="{type}">
      <Distributed>true</Distributed>
      <PreciseAtSecondsLevel>true</PreciseAtSecondsLevel>
      <StartTime>{time}</StartTime>
      <Synchronous>true</Synchronous>
      <TimeUnit ref="{ref}" type="{type}">
      </TimeUnit>
    </Interval>
  </AsynchronousConfiguration>
</Quota>
```

---

#### Elements & Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StartTime</strong> (Mandatory)</td>
<td>Use this attribute only for Quota policies of type calendar. Indicates the date and time when the Quota counter begins counting (regardless of whether or not any requests have been received from any applications.) This value is in UTC. Valid value: date and time, for example 2015-02-09 00:00:00.</td>
</tr>
<tr>
<td><strong>Interval</strong> (Mandatory)</td>
<td>Specifies the interval of time (in hours, minutes, or days as defined by TimeUnit) applicable to the Quota. For example, a value of 24 for the Interval with a TimeUnit of hours means that the Quota is calculated over one day (24 hours). Valid value: integer. The ref attribute identifies the variable that provides the value of the Interval.</td>
</tr>
<tr>
<td><strong>TimeUnit</strong> (Mandatory)</td>
<td>Valid values: second, minute, hour, day, or month. The ref attribute identifies the variable that provides the value of the TimeUnit.</td>
</tr>
<tr>
<td><strong>Allow</strong> (Mandatory)</td>
<td>Specifies the maximum number of inbound requests.</td>
</tr>
<tr>
<td>Elements &amp; Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>count (Optional)</strong></td>
<td>Specifies a message count for the quota.</td>
</tr>
<tr>
<td></td>
<td>For example, a value of 200 for the Allow count with duration of 1 month means that the quota is set to be 200 messages per month.</td>
</tr>
<tr>
<td></td>
<td>Valid value: integer</td>
</tr>
<tr>
<td></td>
<td>Default Value: 2000</td>
</tr>
<tr>
<td><strong>countRef (Optional)</strong></td>
<td>This attribute identifies the variable that provides the value of the Quota limit.</td>
</tr>
<tr>
<td></td>
<td>If a count reference is specified, it takes precedence over the Allow count value.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>The element Allow count=&quot;2000&quot; countRef=&quot;request.header.allowed_quota&quot;/&gt; has a count header (countRef=&quot;request.header.allowed_quota&quot;) along with the count value of 2000.</td>
</tr>
<tr>
<td><strong>Identifier (Optional)</strong></td>
<td>Variable used for uniquely identifying the client application.</td>
</tr>
<tr>
<td></td>
<td>The ref attribute identifies the variable that provides the value of the Identifier.</td>
</tr>
<tr>
<td><strong>MessageWeight (Optional)</strong></td>
<td>Specifies the weight defined for each message.</td>
</tr>
<tr>
<td></td>
<td>Message weight is used to increase impact of request messages that, for example, consume more computational resources than others. For example, you may want to calculate POST messages as being twice as expensive as GET messages.</td>
</tr>
<tr>
<td></td>
<td>A value representing MessageWeight can be extracted from HTTP headers, query parameters, or an XML or JSON request payload.</td>
</tr>
<tr>
<td><strong>Distributed</strong></td>
<td>When set to true, a central counter is maintained that is continuously updated by all API Management servers.</td>
</tr>
<tr>
<td></td>
<td>Note: Always set this value to true.</td>
</tr>
<tr>
<td><strong>PreciseAtSecondsLevel</strong></td>
<td>The default precision for Quotas intervals is one minute. When set to true, Quota precision is set to record at intervals of one second. Use this setting when you have a Quota that uses minutes as the TimeUnit, and you need to ensure that Quotas are counted and enforced by seconds.</td>
</tr>
<tr>
<td></td>
<td>Valid values: true or false</td>
</tr>
<tr>
<td><strong>Synchronous</strong></td>
<td>This setting determines how the distributed Quota counter is updated. If set to true, the quota counter is updated in the central repository synchronously. This means that the update is made at the same time the API call is quota-checked. When synchronous is set to true, you are guaranteed that no API calls over the quota will be allowed.</td>
</tr>
<tr>
<td></td>
<td>Note: Always set this value to true.</td>
</tr>
</tbody>
</table>
Elements & Attributes | Description
---|---
AsynchronousConfiguration(Optional) | This configuration element is only required when Synchronous is set to false. This element enables you to configure the synchronization interval among distributed Quota counters. The default synchronous update interval is 10 seconds. You can modify this by adding the child element SyncIntervalInSeconds. Example

```
  <Synchronous>false</Synchronous>
  <AsynchronousConfiguration>
    <SyncIntervalInSeconds>15</SyncIntervalInSeconds>
  </AsynchronousConfiguration>
```

Alternatively, you can use the SyncMessageCount option instead, but you cannot use both. SyncMessageCount specifies the number of requests across all API Management message processors between quota updates. The following example specifies that the quota count is updated every 10 requests across all API Management message processors:

```
  <Synchronous>false</Synchronous>
  <AsynchronousConfiguration>
    <SyncMessageCount>10</SyncMessageCount>
  </AsynchronousConfiguration>
```

Related Information

- Quota [page 217]
- Reset Quota [page 224]
- Spike Arrest [page 234]

1.5.1.3.4.1.18 Raise Fault

The RaiseFault policy allows you to create custom messages in case of error conditions. This policy returns a FaultResponse to the requesting application if it encounters an error condition.

A FaultResponse can consist of HTTP headers, query parameters, and a message payload. These elements can be populated using variables. This enables you to send customized FaultResponses that are specific to the error conditions.

During execution, the RaiseFault policy transfers the message flow to the default ErrorFlow, which in turn returns the designated FaultResponse to the requesting application. When the message flow switches to the
default ErrorFlow, no further policy processing occurs. All remaining processing steps are bypassed, and the FaultResponse is returned directly to the requesting app.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request →</td>
<td></td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
<tr>
<td></td>
<td>Response</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<!-- The policy will create a custom response of status code as 500 and message as Server error in case of error condition is met -->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RaiseFault async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <FaultResponse>
    <Set>
      <Headers/>
      <Payload contentType="text/plain"> </Payload>
      <StatusCode>500</StatusCode>
      <ReasonPhrase>Server Error</ReasonPhrase>
    </Set>
  </FaultResponse>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</RaiseFault>
```

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnoreUnresolvedVariables(Optional)</td>
<td>Ignores any unresolved variable error in the flow.</td>
</tr>
<tr>
<td></td>
<td>Valid values: true or false</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
</tbody>
</table>

FaultResponse(Optional) Defines the response message returned to the requesting application.

The following predefined flow variables are available after Raise Fault policy executes.

### Flow Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault.name</td>
<td>String</td>
<td>Read-Only</td>
<td>Returns the fault name in the error and if not available, an empty string.</td>
</tr>
<tr>
<td>fault.type</td>
<td>String</td>
<td>Read-Only</td>
<td>Returns the fault type in the error and if not available, an empty string.</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Permission</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fault.category</td>
<td>String</td>
<td>Read-Only</td>
<td>Returns the fault category in the error and if not available, an empty string.</td>
</tr>
</tbody>
</table>

During the policy execution, the following error can occur:

**Error Name** | **Cause**
---|---
RaiseFault | See fault string.

Following fault variables are set when the policy triggers an error at runtime:

**Fault Variables**

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The [prefix] is raisefault. The [policy_name] is the name of the policy that threw the error.</td>
<td>raisefault.RF-ThrowError.failed = true</td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name = ”RaiseFault”</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.19 Reset Quota

The Reset Quota policy enables you to reset the limit for a specified Quota policy.

For example, you can use this policy to reset a Quota counter when additional API calls are to be made. Attach this policy before the Quota policy that you intend to reset.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request →</td>
<td></td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<ResetQuota async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt"/>
```
<Quota name="impose-quota">
  <Identifier ref="request.queryparam.apiKey">
    <Allow>100</Allow>
  </Identifier>
</Quota>
</ResetQuota>

### Elements and Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quota (Mandatory)</strong></td>
<td>Specifies the name of the Quota policy whose counter should be reset.</td>
</tr>
<tr>
<td></td>
<td>The ref attribute identifies the variable that fetches the Quota policy that has to be reset. Both name and ref are optional but at least one of attributes should be used.</td>
</tr>
<tr>
<td><strong>Identifier (Optional)</strong></td>
<td>Specifies the name of the client.</td>
</tr>
<tr>
<td></td>
<td>The ref attribute is the variable used for uniquely identifying the client, for example client_id. Both name and ref are optional but at least one of attributes should be used.</td>
</tr>
<tr>
<td><strong>Allow (Allow integer)</strong></td>
<td>Specifies the message count to which the Quota will be reset.</td>
</tr>
<tr>
<td><strong>Class(Optional)</strong></td>
<td>Refers to the class for which the quota counter will be reset.</td>
</tr>
<tr>
<td></td>
<td>&lt;Class name=&quot;{name}&quot; ref=&quot;request.header.classIdentifier&quot;&gt;</td>
</tr>
</tbody>
</table>

**Reset Quota policy type** defines the following error codes:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidRLPolicyDefinition</td>
<td>Invalid rate limit policy {0}</td>
</tr>
<tr>
<td>NoRLPolicy</td>
<td>Quota policy {0} is not attached.</td>
</tr>
<tr>
<td>InvalidCount</td>
<td>Invalid count value {0} for identifier {1} in {2}</td>
</tr>
<tr>
<td>FailedToResolveAllowCountRef</td>
<td>Failed to resolve allow count reference {0} for identifier {1} in {2}</td>
</tr>
</tbody>
</table>

### Related Information

Quota [page 217]
1.5.1.3.4.1.20 Service Callout

Use this policy to call an external service from your API flow.

A typical scenario consists of the service callout policy, from the response flow, calling a third party API. On receiving a response from the backend service, you (API developer) then call the third party API. The response from this API is then appended to the original response to provide a mashed up response to the requesting application.

While using the Service Callout Policy, it is important to understand the other policies that provision to accomplish a task. The Service Callout is usually used with the Assign Message and Extract Variables policy. The Assign Message policy is used to populate the request message sent to the remote service. The Extract Variables policy is used to parse the response and to extract specific content.

A scenario where you use the three policies is as follows:

1. **Assign Message Policy**: Creates a request message, populates HTTP headers, query parameters, sets the HTTP verb, and so on.
2. **Service Callout Policy**: References a message created by the Assign Message policy, defines a target URL for the external call, and defines a name for the response object that the target service returns.
3. **Extract Variables Policy**: Typically defines a JSONPath or XPath expression that parses the message generated by the preceding Service Callout policy. The policy then sets variables containing the values parsed from the Service Callout response.

You can attach the policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>Preflow</td>
<td>Preflow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ServiceCallout async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <Request/>
  <Response>
    <!-- This policy will call the url api.exampleAPI.com and put the response in variable callOutResponse. -->
    <ServiceCallout async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">  
      <Request/>
      <Response>
        <ServiceCallout>
          <Request/>
          <Response>
            <!-- This policy will call a dynamic url which is set in the previous step via policies like javascript or assign variable. -->
          </Response>
        </ServiceCallout>
      </Response>
    </ServiceCallout>
    <Response>
      <!-- For examples refer the Flow Variables Table.-->  
    </Response>
  </Response>
</ServiceCallout>
```

### Code Syntax

1. <!-- This policy will call the url api.exampleAPI.com and put the response in variable callOutResponse. -->
   - For examples refer the Flow Variables Table. -->
   ```xml
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <ServiceCallout async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
     <Request/>
     <Response>
       <ServiceCallout>
         <Request/>
         <Response>
           <ServiceCallout>
             <Request/>
             <Response>
               <!-- This policy will call a dynamic url which is set in the previous step via policies like javascript or assign variable. -->
             </Response>
           </ServiceCallout>
         </Response>
       </ServiceCallout>
     </Response>
   </ServiceCallout>
   ```
3. <!-- This policy below refers to an existing API Provider. -->
   You can use on-premise, and internet type of API Provider in the Service Callout policy.
   <ServiceCallout async="true" continueOnError="false" enabled="true"
   xmlns="http://www.sap.com/apimgmt">
   <Request/>
   <Response>callOutResponse</Response>
   <Timeout>30000</Timeout>
   <HTTPTargetConnection>
   <APIProvider>API Provider Name</APIProvider>
   <Path>/sap/opu/odata/iwfnd/RMTSAMPLEFLIGHT</Path>
   </HTTPTargetConnection>
   </ServiceCallout>

4 (a). <!-- This policy briefs about SSL Info. SSL stands for Secure Socket Layer. It helps in encrypting the link between a web server and a web client, such as a browser or an app. Although the encrypted link ensures that all data passing between the server and the client remains private. The SSL Info does not support if the API Provider is added to HTTPTargetConnection. Henceforth, there can never be a case where API Provider-SSL Configuration conflicts with SSL Info present in the Service Callout Policy. If enabled = "true" in the code; follow the below code. -->
   <ServiceCallout async="true" continueOnError="false" enabled="true"
   xmlns="http://www.sap.com/apimgmt">
   <Request/>
   <Response>callOutResponse</Response>
   <Timeout>30000</Timeout>
   <HTTPTargetConnection>
   <SSLInfo>
   <Enabled>true</Enabled>
   <ClientAuthEnabled>true</ClientAuthEnabled>
   <KeyStore>SapKeystore</KeyStore>
   <KeyAlias>SAPKey</KeyAlias>
   <TrustStore>SAPTruststore</TrustStore>
   <Ciphers/>
   <Protocols/>
   </SSLInfo>
   </HTTPTargetConnection>
   </ServiceCallout>

4 (b). Use the SSL Info code below, If enabled is set to false.
   <ServiceCallout async="true" continueOnError="false" enabled="false"
   xmlns="http://www.sap.com/apimgmt">
   <Request/>
   <Response>callOutResponse</Response>
   <Timeout>30000</Timeout>
   <HTTPTargetConnection>
   </ServiceCallout>

5. In this Service callout policy, you call a local API Proxy in 2 ways;

5 (a). Using local <APIProxy> and <ProxyEndpoint>
   <ServiceCallout async="true" continueOnError="false" enabled="true"
   xmlns="http://www.sap.com/apimgmt">
   <Request/>
   <Response>callOutResponse</Response>
   <Timeout>30000</Timeout>
   <HTTPTargetConnection>
   <URL>http://{URL_path}</URL>
   </ServiceCallout>

   Note: The protocol example http, https can't be set dynamically.
A callout is typically used with two other policies: Assign Message and Extract Variables.

- **Request**: Assign Message populates the request message sent to the remote service.
- **Response**: Extract Variables parses the response and extracts specific content.

### Elements and Attributes

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Request (Optional)</strong></td>
<td>The variable that contains the request message to be sent by the ServiceCallout.</td>
</tr>
<tr>
<td></td>
<td>- By default, <code>'clearPayload'</code> is false.</td>
</tr>
<tr>
<td></td>
<td>- If <code>'clearPayload'</code> is set to true, the request payload is cleared after the request is sent to the HTTP target.</td>
</tr>
<tr>
<td></td>
<td>- Use the <code>clearPaylo</code> option only if the request message is not required after the Service Callout is executed, because <code>clearPayload</code> allocates memory during message processing.</td>
</tr>
<tr>
<td></td>
<td>- The policy returns an error if the request message cannot be resolved by the element or it is of an invalid request message type.</td>
</tr>
<tr>
<td>Elements and Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Response (Optional)</td>
<td>Output of the ServiceCallout (usually the response message received from the target) that will be assigned to the response variable.</td>
</tr>
<tr>
<td></td>
<td>The output generated by the policy is assigned to the variable only when the entire response is read successfully by the policy. If the response message fails for any reason, the policy returns an error.</td>
</tr>
<tr>
<td></td>
<td>If this element is not specified, the policy execution does not wait for response to be completely read and executes the message flow steps.</td>
</tr>
<tr>
<td>Timeout (Optional)</td>
<td>The time in milliseconds that the ServiceCallout policy will wait for a response from the target before exiting. The default timeout for ServiceCallout is determined by the default HTTP timeout setting for API Management, which is 55000 milliseconds (55 seconds).</td>
</tr>
</tbody>
</table>
### Elements and Attributes

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPTargetConnection</td>
<td>Provides transport details such as URL and HTTP properties.</td>
</tr>
</tbody>
</table>

**Note**

You can use flow variables to construct the URL in an HTTPTargetConnection element.

#### <HTTPTargetConnection>/<URL> element:

You can supply portion of the URL that changes with a variable. Although, the protocol part of the URL, http:// beneath, cannot be stated by a variable. In the next example, you use a variable to emphasize the value of query parameter.

```
<URL>http://example.com/forecastrss?w=${request.header.woeid}</URL>
```

Or, set a part of the URL path by a variable:

```
<URL>http://example.com/{request.resourcePath}?w=${request.header.woeid}</URL>
```

If you need to use a variable to quantify the domain and port of the URL, then use one variable alone for the domain and port, and a second variable for the other portion of the URL:

```
<URL>http://{request.dom_port}/{request.resourcePath}</URL>
```

Refer sample code (1)

#### <LocalTargetConnection> element:

It quantifies a local proxy within the environment and acts as the target of service callouts.

#### <LocalTargetConnection>/<ProxyEndpoint> element

It is a proxy endpoint in the API proxy quantified with the <APIProxy> element.

Refer sample code 4 (a)

#### <LocalTargetConnection>/<Path> element

It targets the path to the endpoint. The endpoint must refer to a local proxy while making the call.

Refer sample code 4 (b).

### Flow variables

They allow dynamic performance of policies and flows at runtime. It’s based on HTTP headers, message content, or Flow context. The following Flow variables are available and predefined after a Service Callout policy is executed.
Following is an example of getting Service Callout request and response headers similar to how you would get headers from the main request and response.

- **calloutResponse.header.HeaderName**
  - Scope: From the Service Callout forward
  - Type: String
  - Permission: Read/Write
  - A message header in the Service Callout request or response

- **myRequest.header.HeaderName**
  - Note: The name here is for the request.

where calloutResponse is the variable name for the Response in the Service Callout, and myRequest is the variable name for the Request. For example:

- **calloutResponse.header.Content-Length**
  - returns the Content-Length header of the Service Callout response.

---

**servicecallout.requesturi**
- Scope: From the Service Callout request forward
- Type: String
- Permission: Read/Write
- The TargetEndpoint URI for a ServiceCallout policy. The URI is the TargetEndpoint URL without the protocol and domain specification.

**servicecallout.{policy-name}.target.url**
- Scope: From the Service Callout request forward
- Type: String
- Permission: Read/Write
- The target URL for the Service Callout.

**calloutResponse.content**
- Note: The variable name here is in the Service Callout configuration.
- Scope: From the Service Callout response forward
- Type: String
- Permission: Read/Write
- The response body from the Service Callout.

**servicecallout.{policy-name}.expectedcn**
- Note: The variable name here is in the Service Callout configuration.
- Scope: From the Service Callout request forward
- Type: String
- Permission: Read/Write
- The expected Common Name of the TargetEndpoint as referred to in a ServiceCallout policy. This is meaningful only when the TargetEndpoint refers to an TLS/SSL endpoint.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| servicecallout.{policy-name}.failed | Scope: From the Service Callout response forward  
| | Type: Boolean  
| | Permission: Read/Write  
| | Boolean indicating if the policy succeeded, false, or failed, true. |

**Errors**

This segment defines the fault codes and error messages that are returned.

During the policy execution, the following errors can occur:

**Error Code**

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestVariableNotMessageType</td>
<td>The Request variable specified in the policy is not of type Message. For example, if it’s a string or other non-message type, you’ll see this error.</td>
</tr>
<tr>
<td>RequestVariableNotRequest\MessageType</td>
<td>The Request variable specified in the policy is not of type Request Message. For example, if it’s a Response type, you’ll see this error.</td>
</tr>
</tbody>
</table>
| ExecutionFailed | This error can occur when:  
| | ● The policy is asked to handle input that is malformed or otherwise invalid.  
| | ● The backend target service returns an error status (by default, 4xx or 5xx). |
| ErrorResponseCode | The backend target service returns an error status (by default, 4xx or 5xx). |
| InvalidExecutionState | JSONThreatProtection[policy_name]: Exceeded string value length at line [line_num] |
| URLMissing | The <URL> element inside <HTTPTargetConnection> is missing or empty. |
| ConnectionInfoMissing | This error happens if the policy does not have an <HTTPTargetConnection> element. |
| InvalidTimeoutValue | This error happens if the <Timeout> value is negative or zero. |

**Runtime Errors**

<table>
<thead>
<tr>
<th>Fault Code</th>
<th>Cause</th>
</tr>
</thead>
</table>
| steps.servicecallout.Executionfailed | This error can arise when:  
| | The policy is requested to handle input that is distorted or otherwise unacceptable. |
Fault Code | Cause
---|---
steps.servicecallout.RequestVariableNotMessageType | The Request variable is quantified in the policy and is not of type Message. For instance, if it's a string or other non-message type, you'll encounter this error.
steps.servicecallout.RequestVariableNotRequestMessageType | The Request variable quantified in the policy is not of type Request Message. For instance, if it's a Response type, you'll encounter this error.

**sample error response**

```json
{
  "fault": {
    "detail": {
      "errorcode": "steps.servicecallout.RequestVariableNotMessageT ype",
      "faultstring": "ServiceCallout[ServiceCalloutGetMockResponse]: request variable data_str value is not of type Message"
    }
  }
}
```

**Sample fault rule**

```xml
<faultrule name="VariableOfNonMsgType"></faultrule>
<FaultRule name="RequestVariableNotMessageType">
  <Step>
    <Name>AM-RequestVariableNotMessageType</Name>
  </Step>
  <Condition>(fault.name = "RequestVariableNotMessageType")</Condition>
</FaultRule>
```

Following fault variables are set when the policy triggers an error at runtime:

**Fault Variables**

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The [prefix] is servicecallout. [policy_name]: The name of the policy to check.</td>
<td>servicecallout.SC-GetUserData.failed = true</td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name = “RequestVariableNotMessageType”</td>
</tr>
</tbody>
</table>

**Related Information**

Assign Message [page 104]
1.5.1.3.4.1.21 Spike Arrest

The Spike Arrest policy limits the number of requests forwarded from the point in the processing flow where the policy is attached as a processing step.

You can attach a Spike Arrest policy at the proxy endpoint or the target endpoint. At the proxy endpoint, this policy limits inbound requests. When you attach this policy at the TargetEndpoint, it limits request forwarded to the backend service.

Unlike Quotas, spike arrests are not implemented as counts. Rather, they are implemented as a rate limit which is based on the time the last matching message was processed. If you specify 6 messages per minute, it means that requests can only be submitted at the rate of one per 10 second interval. A second request within 10 seconds on the same API Management server will be rejected. Even with a larger number (200 per second), if two requests come in nearly simultaneously to the same API Management server, one will be rejected. Each successful request will update the spike arrest’s last processed count.

No counter is maintained for spike arrests, only a time that the last message was successfully passed through the Spike Arrest policy. On a given API Management server, if a request is received now, all subsequent requests will fail until 10 seconds has elapsed. Since Spike Arrest is not distributed, you might see some discrepancy between the actual behavior of the system and your expected results. In general, you should use Spike Arrest to set a limit that throttles traffic to what your backend services can handle. Do not use Spike Arrest to limit traffic from individual clients.

You can attach the policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td></td>
</tr>
<tr>
<td>PreFlow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>Flow</td>
</tr>
<tr>
<td>Flow</td>
<td></td>
</tr>
<tr>
<td>PreFlow</td>
<td>Flow</td>
</tr>
<tr>
<td>Flow</td>
<td></td>
</tr>
<tr>
<td>PostFlow</td>
<td>Flow</td>
</tr>
<tr>
<td>Flow</td>
<td></td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SpikeArrest async="true" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <Identifier ref="request.header.userid"></Identifier>
  <MessageWeight ref="request.header.weight"></MessageWeight>
  <Rate>30pm</Rate>
</SpikeArrest>
```
<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier ref</td>
<td>Variable used for uniquely identifying the application or client.</td>
</tr>
<tr>
<td>MessageWeight ref</td>
<td>Specifies the weight defined for each message. Message weight is used to modify the impact of a single request on the calculation of the SpikeArrest limit. Message weight can be set by variables based on HTTP headers, query parameters, or message body content. For example, if the SpikeArrest Rate is 10 per minute, and an application submits requests with weight 5, then only 2 messages are permitted per minute from that application.</td>
</tr>
<tr>
<td>Rate</td>
<td>Specifies the rate at which to limit the traffic spike (or burst). Valid value: integer per &lt;min&gt; or &lt;sec&gt; or &lt;variable&gt;.</td>
</tr>
</tbody>
</table>

When a Spike Arrest policy executes, the following Flow variable is populated:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ratelimit.[policy_name].failed</td>
<td>Boolean</td>
<td>Read-Only</td>
<td>Indicates whether or not the policy failed (true or false).</td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>HTTP Status</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpikeArrestViolation</td>
<td>500</td>
<td>The rate limit is exceeded.</td>
</tr>
<tr>
<td>InvalidMessageWeight</td>
<td>500</td>
<td>The message weight value must be an integer.</td>
</tr>
<tr>
<td>FailedToResolveSpikeArrestRate</td>
<td>500</td>
<td>The referenced variable used to specify the rate can't be resolved.</td>
</tr>
</tbody>
</table>

Following fault variables are set when the policy triggers an error at runtime:

<table>
<thead>
<tr>
<th>Fault Variables</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The fault variable [prefix] is ratelimit. The [policy_name] is the name of the policy that threw the error.</td>
<td>ratelimit.SA-SpikeArrestPolicy.failed = true</td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td>fault.name Matches &quot;SpikeArrestViolation&quot;</td>
</tr>
</tbody>
</table>

**Related Information**

*Quota [page 217]*
OAuth 2.0 defines an authorization protocol for protected API resources.

To ensure that applications are allowed to act on behalf of users, OAuth 2.0 relies on ‘access tokens’. To access protected resources, consumer applications must obtain ‘access tokens’. The OAuth 2.0 specification defines the various ways that applications can request and use access tokens. API Management provides a policy type that enables you to configure OAuth 2.0 authorization for your APIs.

Setting up OAuth 2.0 authorization for your API is a three step process:

1. Configure a token endpoint: An OAuth token endpoint defines a URI on API Management. The token endpoint is configured with a policy of type OAuthV2. In the OAuthV2 policy, the GenerateAccessToken operation is specified. When this operation is specified, you have the option of configuring one or more grant types. For each grant type specified, an additional set of configuration elements are exposed, providing flexibility in the way that APIs exposed through API Management manage OAuth-based authorization.

2. Apply an OAuth validation policy to protected resource URIs: To enforce OAuth at runtime, attach a policy of type OAuthV2 to a Flow that exposes a protected resource. In the OAuthV2 policy, specify the VerifyAccessToken operation.

3. Configure one or more API products: The VerifyAccessToken operation resolves the access token to an API product for which the application has been approved. The request URI is verified against the list of URIs defined in the API product. If the request URI is included in the list defined by the approved API product, then the request is forwarded to the protected resource.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th></th>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>＞ ＿ ＿ ＿ ＿</td>
<td>＿ ＿ ＿</td>
</tr>
<tr>
<td></td>
<td>＿ ＿ ＿ ＿</td>
<td>＿ ＿ ＿</td>
</tr>
<tr>
<td></td>
<td>＿ ＿ ＿ ＿</td>
<td>＿ ＿ ＿</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessToken</td>
<td>By default, VerifyAccessToken expects the access token to be sent in an Authorization header. You can change that default using this element. For example request.queryparam.access_token indicates that the access token should be present as a query parameter.</td>
</tr>
<tr>
<td>AppEndUser</td>
<td>This element lets you specify where API Management should look for the end user ID</td>
</tr>
<tr>
<td>ClientId</td>
<td>This element lets you specify where API Management should look for the end user ID</td>
</tr>
<tr>
<td><strong>Elements &amp; Attributes</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>Code</strong></td>
<td>This element lets you specify where API Management should look for the authorization code. For example, it could be sent as a query parameter, HTTP header, or form parameter (the default).</td>
</tr>
<tr>
<td><strong>ExpiresIn</strong></td>
<td>Enforces the expiry time of access tokens and authorization codes in milliseconds. (For refresh tokens, use <code>&lt;RefreshTokenExpiresIn&gt;</code>.) The expiry time value is a system generated value plus the <code>&lt;ExpiresIn&gt;</code> value. If <code>&lt;ExpiresIn&gt;</code> is set to -1, the token or code is given an infinite lifetime. If <code>&lt;ExpiresIn&gt;</code> is not specified, the system applies a default value configured at the system level.</td>
</tr>
<tr>
<td><strong>ExternalAccessToken</strong></td>
<td>Tells API Management where to find an external access token</td>
</tr>
<tr>
<td><strong>GenerateResponse</strong></td>
<td>If set to true, the policy generates and returns a response</td>
</tr>
<tr>
<td><strong>GenerateErrorResponse</strong></td>
<td>If set to true, the policy generates and returns a response if the ContinueOnError attribute is set to true. If false (the default), no response is sent.</td>
</tr>
<tr>
<td><strong>GrantType</strong></td>
<td>Tells the policy where to find the grant type parameter that is passed in a request.</td>
</tr>
<tr>
<td><strong>RedirectUri</strong></td>
<td>Specifies where to should look for the redirect_uri parameter in the request.</td>
</tr>
<tr>
<td><strong>RefreshToken</strong></td>
<td>When requesting an access token using a refresh token, you must supply the refresh token in the request. This element lets you specify where API Management should look for the refresh token. For example, it could be sent as a query parameter, HTTP header, or form parameter.</td>
</tr>
<tr>
<td><strong>RefreshTokenExpiresIn</strong></td>
<td>Enforces the expiry time of refresh tokens in milliseconds. The expiry time value is a system generated value plus the <code>&lt;RefreshTokenExpiresIn&gt;</code> value. If <code>&lt;RefreshTokenExpiresIn&gt;</code> is set to -1, the refresh token is given an infinite lifetime. If <code>&lt;RefreshTokenExpiresIn&gt;</code> is not specified, the system applies a default value configured at the system level.</td>
</tr>
<tr>
<td><strong>ResponseType</strong></td>
<td>This element informs API Management which grant type the client app is requesting. It is used only with the authorization code and implicit grant type flows.</td>
</tr>
<tr>
<td><strong>ReuseRefreshToken</strong></td>
<td>When set to true, the existing refresh token is reused until it expires. If false, a new refresh token is issued by API Management when a valid refresh token is presented.</td>
</tr>
<tr>
<td><strong>PassWord</strong></td>
<td>This element is used with the password grant type only. With the password grant type, user credentials (password and username) must be made available to the OAuthV2 policy. The <code>&lt;PassWord&gt;</code> and <code>&lt;UserName&gt;</code> elements are used to specify variables where API Management can find these values. If these elements are not specified, the policy expects to find the values (by default) in form parameters named username and password.</td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>If this element is present in one of the GenerateAccessToken or GenerateAuthorizationCode policies, it is used to specify which scopes to grant the token or code.</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>In cases where the client app must send the state information to the authorization server, this element lets you specify where API Management should look for the state values. For example, it could be sent as a query parameter or in an HTTP header.</td>
</tr>
</tbody>
</table>
Elements & Attributes | Description
---|---
StoreToken | Set this element to true when the <ExternalAuthorization> element is true. The <StoreToken> element tells API Management to store the external access token. Otherwise, it will not be persisted.

Following flow variables are populated when the policy is executed:

Flow Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>organization_name</td>
<td>The name of the organization where the proxy is executing.</td>
</tr>
<tr>
<td>developer.id</td>
<td>The ID of the developer associated with the registered client app.</td>
</tr>
<tr>
<td>developer.app.name</td>
<td>The name of the developer associated with the registered client app.</td>
</tr>
<tr>
<td>client_id</td>
<td>The client ID of the registered client app.</td>
</tr>
<tr>
<td>grant_type</td>
<td>The grant type associated with the request.</td>
</tr>
<tr>
<td>token_type</td>
<td>The token type associated with the request.</td>
</tr>
<tr>
<td>access_token</td>
<td>The access token that is being verified.</td>
</tr>
<tr>
<td>accesstoken.({custom_attribute})</td>
<td>A named custom attribute in the access token.</td>
</tr>
<tr>
<td>issued_at</td>
<td>The date the access token was issued.</td>
</tr>
<tr>
<td>expires_in</td>
<td>The expiration time for the access token</td>
</tr>
<tr>
<td>status</td>
<td>The status of the access token (for example, approved or revoked).</td>
</tr>
<tr>
<td>scope</td>
<td>The scope (if any) associated with the access token</td>
</tr>
<tr>
<td>apiproduct.({custom_attribute_name})</td>
<td>A named custom attribute of the API product associated with the registered client app.</td>
</tr>
<tr>
<td>apiproduct.name</td>
<td>The name of the API product associated with the registered client app.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>App-specific variables</td>
<td></td>
</tr>
<tr>
<td>app.name</td>
<td></td>
</tr>
<tr>
<td>app.id</td>
<td></td>
</tr>
<tr>
<td>app.accessType</td>
<td></td>
</tr>
<tr>
<td>app.callbackUrl</td>
<td></td>
</tr>
<tr>
<td>app.status</td>
<td></td>
</tr>
<tr>
<td>app.scopes</td>
<td></td>
</tr>
<tr>
<td>app.appFamily</td>
<td></td>
</tr>
<tr>
<td>app.apiproducts</td>
<td></td>
</tr>
<tr>
<td>app.appParentStatus</td>
<td></td>
</tr>
<tr>
<td>app.appType</td>
<td></td>
</tr>
<tr>
<td>app.appParentId</td>
<td></td>
</tr>
<tr>
<td>app.created_by</td>
<td></td>
</tr>
<tr>
<td>app.created_at</td>
<td></td>
</tr>
<tr>
<td>app.last_modified_at</td>
<td></td>
</tr>
<tr>
<td>app.last_modified_by</td>
<td></td>
</tr>
<tr>
<td>app.{custom_attributes}</td>
<td></td>
</tr>
<tr>
<td>Developer-specific variables</td>
<td></td>
</tr>
<tr>
<td>developer.id</td>
<td></td>
</tr>
<tr>
<td>developer.userName</td>
<td></td>
</tr>
<tr>
<td>developer.firstName</td>
<td></td>
</tr>
<tr>
<td>developer.lastName</td>
<td></td>
</tr>
<tr>
<td>developer.email</td>
<td></td>
</tr>
<tr>
<td>developer.status</td>
<td></td>
</tr>
<tr>
<td>developer.apps</td>
<td></td>
</tr>
<tr>
<td>developer.created_by</td>
<td></td>
</tr>
<tr>
<td>developer.created_at</td>
<td></td>
</tr>
<tr>
<td>developer.last_modified_at</td>
<td></td>
</tr>
<tr>
<td>developer.last_modified_by</td>
<td></td>
</tr>
<tr>
<td>developer.{custom_attributes}</td>
<td></td>
</tr>
</tbody>
</table>

Note: If the app.appType is "Developer", then developer attributes are populated.

During the policy execution, the following errors can occur:
## Error Cause

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidClientIdentifier</td>
<td>The client identifier sent from the client is invalid or missing. Check to be sure you are using the correct client key and secret values for the Developer App associated with your proxy. Typically, these values are sent as a Base64 encoded Basic Authorization header. Important: This error name used to be called invalid_client.</td>
</tr>
<tr>
<td>invalid_client</td>
<td>This error name is no longer used. It was replaced by InvalidClientIdentifier.</td>
</tr>
<tr>
<td>invalid_request</td>
<td>This error name is used for multiple different kinds of errors, typically for missing or incorrect parameters sent in the request. If &lt;GenerateResponse&gt; is set to false, use fault variables (described below) to retrieve details about the error, such as the fault name and cause.</td>
</tr>
<tr>
<td>invalid_access_token</td>
<td>The access token sent from the client is invalid.</td>
</tr>
<tr>
<td>FailedToResolveToken</td>
<td>The policy expected to find a token in a variable specified in the &lt;Tokens&gt; element, but the variable could not be resolved.</td>
</tr>
<tr>
<td>FailedToResolveClientId</td>
<td>The policy expected to find the Client ID in a variable specified in the &lt;ClientId&gt; element, but the variable could not be resolved.</td>
</tr>
<tr>
<td>FailedToResolveAccessToken</td>
<td>The policy expected to find an access token in a variable specified in the &lt;AccessToken&gt; element, but the variable could not be resolved.</td>
</tr>
<tr>
<td>FailedToResolveRefreshToken</td>
<td>The policy expected to find a refresh token in a variable specified in the &lt;RefreshToken&gt; element, but the variable could not be resolved.</td>
</tr>
<tr>
<td>FailedToResolveAuthorizationCode</td>
<td>The policy expected to find an authorization code in a variable specified in the &lt;Code&gt; element, but the variable could not be resolved.</td>
</tr>
<tr>
<td>UnSupportedGrantType</td>
<td>The client specified a grant type that is unsupported by the policy</td>
</tr>
<tr>
<td>InvalidTokenType</td>
<td>The &lt;Tokens&gt;/&lt;Token&gt; element requires you to specify the token type (for example, refreshToken). If the client passes the wrong type, this error is thrown.</td>
</tr>
<tr>
<td>InvalidAPICallAsNoApiProductMatchFound</td>
<td>The API proxy is not in the Product associated with the access token.</td>
</tr>
<tr>
<td>InsufficientScope</td>
<td>The access token presented in the request has a scope that does not match the scope specified in the verify access token policy.</td>
</tr>
<tr>
<td>InvalidParameter</td>
<td>The policy must specify either an access token or an authorization code, but not both.</td>
</tr>
<tr>
<td>MissingParameter</td>
<td>The response type is token, but no grant types are specified.</td>
</tr>
<tr>
<td>InvalidValueForExpiresIn</td>
<td>For the &lt;ExpiresIn&gt; element, valid values are positive integers and -1.</td>
</tr>
<tr>
<td>InvalidValueForRefreshTokenExpiresIn</td>
<td>For the &lt;RefreshTokenExpiresIn&gt; element, valid values are positive integers and -1.</td>
</tr>
<tr>
<td>InvalidGrantType</td>
<td>An invalid grant type is specified in the &lt;SupportedGrantTypes&gt; element.</td>
</tr>
<tr>
<td>ExpiresInNotApplicableForOperation</td>
<td>Be sure that the operations specified in the &lt;Operations&gt; element support expiration.</td>
</tr>
<tr>
<td>RefreshTokenExpiresInNotApplicableForOperation</td>
<td>Be sure that the operations specified in the &lt;Operations&gt; element support refresh token expiration.</td>
</tr>
<tr>
<td>Error Name</td>
<td>Cause</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GrantTypesNotApplicableForOperation</td>
<td>Be sure that the grant types specified in <code>&lt;SupportedGrantTypes&gt;</code> are supported for the specified operation.</td>
</tr>
<tr>
<td>OperationRequired</td>
<td>You must specify an operation in this policy using the <code>&lt;Operation&gt;</code> element.</td>
</tr>
<tr>
<td>InvalidOperation</td>
<td>You must specify a valid operation in this policy using the <code>&lt;Operation&gt;</code> element.</td>
</tr>
<tr>
<td>TokenValueRequired</td>
<td>You must specify a token <code>&lt;Token&gt;</code> value in the <code>&lt;Tokens&gt;</code> element.</td>
</tr>
</tbody>
</table>

Following fault variables are set when the policy triggers an error at runtime:

**Fault Variables**

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[prefix].[policy_name].failed</code></td>
<td>The <code>[prefix]</code> is oauthV2. The <code>[policy_name]</code> is the name of the policy that threw the error.</td>
<td><code>oauthV2.GenerateAccessToken.failed = true</code></td>
</tr>
<tr>
<td><code>[prefix].[policy_name].fault.name</code></td>
<td>The <code>[prefix]</code> is oauthV2. The <code>[policy_name]</code> is the name of the policy that threw the error.</td>
<td><code>oauthV2.GenerateAccessToken.fault.name = invalid_request</code></td>
</tr>
<tr>
<td>Note: For the VerifyAccessToken operation, the fault name includes this suffix: keymanagement.service. For example: keymanagement.service.invalid_access_token</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>[prefix].[policy_name].fault.cause</code></td>
<td>The <code>[prefix]</code> is oauthV2. The <code>[policy_name]</code> is the name of the policy that threw the error.</td>
<td><code>oauthV2.GenerateAccessToken.cause = Required param: grant_type</code></td>
</tr>
<tr>
<td><code>fault.name = [error_name]</code></td>
<td><code>[error_name]</code> is the specific error name to check for as listed in the table above.</td>
<td><code>fault.name = &quot;invalid_request&quot;</code></td>
</tr>
</tbody>
</table>

**Related Information**

- Generate Access Token [page 242]
- Generate Authorization Code [page 243]
- Verify Access Tokens [page 244]
- Designing OAuth v2.0 Policies [page 245]
- OAuth 2.0 Grant Types [page 248]
1.5.1.3.4.1.22.1 Generate Access Token

OAuth 2.0 policies are used both to generate and to validate OAuth 2.0-compliant tokens. To generate tokens on behalf of application end users, OAuth 2.0 policies that specify the GenerateAccessToken operation are attached to a token endpoint.

**Note**

Deploy a single API proxy to function as a token endpoint for all API proxies in an environment. A single API proxy configured as a token endpoint can support multiple grant types. By setting up a single token endpoint, you can publish a unified set of URIs that application developers can use to obtain tokens.

A token endpoint is simply a URI path that the system uses to identify requests for access tokens. On API Management, a token endpoint is a conditional flow to which an OAuthV2 policy is attached. The OAuthV2 policy specifies the GenerateAccessToken operation as an element. For example, to configure a token endpoint that generates tokens on requests to the URI path /accesstoken:

**Sample Code**

```xml
<Flow name="TokenEndpoint">
  <Condition>proxy.pathsuffix MatchesPath "/accesstoken"</Condition>
  <Step><Name>GenerateAccessToken</Name></Step>
</Flow>
```

**Note**

An example payload for the policy is as follows.

**Code Syntax**

```xml
<OAuthV2 async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <!-- this flag has to be set when you want to work with third-party access tokens -->
  <ExpiresIn ref="kvm.expiry.value">360000</ExpiresIn> <!-- in mili seconds -->
  <ExternalAuthorization>false</ExternalAuthorization>
  <GrantType>client_credentials</GrantType>
  <Operation>GenerateAccessToken</Operation>
  <GenerateResponse enabled="true"/>
  <SupportedGrantTypes>
    <GrantType>client_credentials</GrantType>
  </SupportedGrantTypes>
</OAuthV2>
```

These variables are set when the GenerateAccessToken policy operation executes successfully for the authorization code, password, and client credentials grant type flows. Note that refresh token variables do not apply to and are not set by the client credentials grant type flow.
### Access Token

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oauthv2accesstoken.{policy_name}.token_type</td>
<td>Will be set to accesstoken.</td>
</tr>
<tr>
<td>oauthv2accesstoken.{policy_name}.expires_in</td>
<td>The expiry value for the token.</td>
</tr>
<tr>
<td>oauthv2accesstoken.{policy_name}.refresh_token</td>
<td>The refresh token generated when the policy executes.</td>
</tr>
<tr>
<td>oauthv2accesstoken.{policy_name}.refresh_token_expires_in</td>
<td>The lifespan of the refresh token, in seconds.</td>
</tr>
<tr>
<td>oauthv2accesstoken.{policy_name}.refresh_token_issued_at</td>
<td>This time value is the string representation of the corresponding 32-bit timestamp quantity.</td>
</tr>
<tr>
<td>oauthv2accesstoken.{policy_name}.refresh_token_status</td>
<td>Set to approved or revoked.</td>
</tr>
<tr>
<td>oauthv2accesstoken.{policy_name}.scope</td>
<td>List of available OAuth scopes.</td>
</tr>
</tbody>
</table>

For information on the various GrantType supported, see OAuth 2.0 Grant Types [page 248]. For information on the various field descriptions (supported elements and attributes), see Designing OAuth v2.0 Policies [page 245].

### Related Information

- Designing OAuth v2.0 Policies [page 245]
- OAuth 2.0 Grant Types [page 248]

### 1.5.1.3.4.1.22.2 Generate Authorization Code

Similarly, you can configure authorization endpoints to issue authorization codes. For example:

```xml
<Flow name="AuthorizationEndpoint">
  <Condition>proxy.pathsuffix == "/authorize"</Condition>
  <Request>
    <Step><Name>GenerateAuthCode</Name></Step>
  </Request>
</Flow>
```

This policy needs to be attached to the response to generate the authorization code.
An example payload for the policy is as follows.

```xml
<OAuthV2 async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
    <ClientId>request.queryparam.client_id</ClientId>
    <Operation>GenerateAuthorizationCode</Operation>
    <RedirectUri>request.queryparam.redirect_uri</RedirectUri>
    <GenerateResponse enabled="true"/>
    <ResponseType>request.queryparam.response_type</ResponseType>
    <Scope>request.queryparam.scope</Scope>
</OAuthV2>
```

Here the response will be HTTP 302 and the code will be sent to app-redirect-url as an query parameter e.g.

```
HTTP 302 https://<your-app-redirect-url>?code=<the_generate_code>
&state=1&scope=read
```

These variables are set when the GenerateAuthorizationCode policy operation executes successfully:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oauthv2authcode.{policy_name}.code</code></td>
<td>The authorization code generated when the policy executes.</td>
</tr>
<tr>
<td><code>oauthv2authcode.{policy_name}.redirect_uri</code></td>
<td>The redirect URI associated with the registered client app.</td>
</tr>
<tr>
<td><code>oauthv2authcode.{policy_name}.scope</code></td>
<td>The optional OAuth scope passed in the client request.</td>
</tr>
<tr>
<td><code>oauthv2authcode.{policy_name}.client_id</code></td>
<td>The client ID passed in the client request.</td>
</tr>
</tbody>
</table>

For information on the various GrantType supported, see OAuth 2.0 Grant Types [page 248]. For information on the various field descriptions (supported elements and attributes), see Designing OAuth v2.0 Policies [page 245].

### 1.5.1.3.4.1.22.3 Verify Access Tokens

Once a token endpoint is set up for an API proxy, a corresponding OAuthV2 policy that specifies the `VerifyAccessToken` operation is attached to the Flow that exposes the protected resource.

**Example**

To ensure that all requests to an API are authorized, the following policy enforces access token verification:
The policy is attached to the API resource to be protected. To ensure that all requests to an API are verified, attach the policy to the proxy endpoint request PreFlow, as follows:

**Sample Code**

```xml
<PreFlow>
  <Request>
    <Step><Name>VerifyOAuthAccessToken</Name></Step>
  </Request>
</PreFlow>
```

The following optional elements can be used to override the default settings for the VerifyAccessToken operation:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>A space separated list of scopes that must be present in the access token for verification to succeed. For example, the following policy will check the access token to ensure that it contains the scopes READ and WRITE:</td>
</tr>
<tr>
<td></td>
<td><strong>Sample Code</strong></td>
</tr>
</tbody>
</table>
|            | `<OAuthV2>  
|            |  <Operation>VerifyAccessToken</Operation>  
|            |  </OAuthV2>` |
| AccessToken | The variable where the access token is expected to be located. For example, request.queryparam.accesstoken. By default, the access token is expected to be presented by the application in the Authorization HTTP header, according to the OAuth 2.0 specification. Use this setting if the access token is expected to be presented in a nonstandard location. Such location may be a query parameter, or an HTTP header with a name other than Authorization. |

For information on the various field descriptions (supported elements and attributes), see Designing OAuth v2.0 Policies [page 245].

### 1.5.1.3.4.1.22.4 Designing OAuth v2.0 Policies

The table below illustrates the various elements and attributes used in the OAuth policies:
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Valid Values</th>
<th>Related Operation and Grant Type</th>
<th>Combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>The OAuth 2.0 operation implemented by the policy</td>
<td>GenerateAccessToken</td>
<td>GenerateAccessTokenImplicit-Grant</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GenerateAuthorizationCode</td>
<td>RefreshAccessToken</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VerifyAccessToken</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExpiresIn(optional)</td>
<td>ExpiresIn enforces the expiry time of the authorization code in milliseconds. The expiry time of authorization code is system generated plus ExpiresIn value. If ExpiresIn is -1, the system considers it as an infinite life time. If it is not specified, the system applies a default value configured at system level.</td>
<td>GenerateAccessToken</td>
<td>GenerateAccessTokenImplicit-Grant</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GenerateAuthorizationCode</td>
<td>RefreshAccessToken</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VerifyAccessToken</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SupportedGrant Types</td>
<td>Specifies the GrantTypes supported by a token endpoint. An endpoint may support multiple GrantTypes (that is, a single endpoint can be configured to distribute access tokens for multiple GrantTypes.)</td>
<td>client_credentials</td>
<td></td>
<td>All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>authorization_code</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>implicit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>The expected location in the request message where the authorization code must be presented to the token endpoint</td>
<td>Any variable setting. For example request.queryparam.auth_code indicates that the authorization code should be present as a query parameter, as, for example, ?auth_code=AfGlvs9. To require the authorization code in an HTTP header, for example, set this value to request.header.auth_code.</td>
<td>GenerateAccessToken with grant type authorization_code</td>
<td>GenerateAccessToken with</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Valid Values</td>
<td>Related Operation and Grant Type</td>
<td>Combinations</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>--------------</td>
<td>----------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>ClientId</td>
<td>The expected location in the request message where the client_id (the app’s consumer key) must be presented to the token endpoint</td>
<td>Any variable setting. For example request.queryparam.client_id indicates that the client_id should be present as a query parameter, as, for example, ?client_id=AfGlvs9. To require the ClientId in an HTTP header, for example, set this value to request.header.client_id.</td>
<td>GenerateAccessToken</td>
<td>Implicit: Optional, GenerateAuthorization: Code:Optional</td>
</tr>
<tr>
<td>RedirectUri</td>
<td>The expected location in the request message where the RedirectUri must be presented to the token endpoint.</td>
<td>Any variable setting. For example request.queryparam.redirect_uri indicates that the RedirectUri should be present as a query parameter, as, for example, ?redirect_uri=login.myapp.com. To require the RedirectUri in an HTTP header, for example, set this value to request.header.redirect_uri.</td>
<td>GenerateAccessToken</td>
<td>Implicit: Optional, GenerateAuthorization: Code:Optional</td>
</tr>
<tr>
<td>Scope</td>
<td>The expected location in the request message where the scope must be presented to the token endpoint.</td>
<td>Any variable setting. For example request.queryparam.scope indicates that the scope should be present as a query parameter, as, for example, ?scope=READ. To require the scope in an HTTP header, for example, set this value to request.header.scope.</td>
<td>GenerateAccessToken</td>
<td>All: Optional</td>
</tr>
<tr>
<td>State</td>
<td>The expected location in the request message where the state must be presented to the token endpoint.</td>
<td>Any variable setting. For example request.queryparam.state indicates that the state should be present as a query parameter, as, for example, ?state=HjoiuKJH32. To require the state in an HTTP header, for example, set this value to request.header.state.</td>
<td>GenerateAccessToken</td>
<td>All: Optional, authorization_code, password</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Valid Values</td>
<td>Related Operation and Grant Type</td>
<td>Combinations</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>AppEndUser</td>
<td>The expected location in the request message where the state must be presented to the token endpoint.</td>
<td>Any variable setting. For example request.queryparam.app_enduser indicates that the AppEndUser should be present as a query parameter, as, for example, ?app_enduser=<a href="mailto:ntesla@theramin.com">ntesla@theramin.com</a>. To require the AppEndUser in an HTTP header, for example, set this value to request.header.app_enduser.</td>
<td>All: Optional</td>
<td></td>
</tr>
<tr>
<td>UserName</td>
<td>The expected location in the request message where the UserName must be presented to the token endpoint.</td>
<td>Any variable setting. For example request.queryparam.username indicates that the username should be present as a query parameter, as, for example, ?username=joe. To require the UserName in an HTTP header, for example, set this value to request.header.username.</td>
<td>All: Optional</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>The expected location in the request message where the Password must be presented to the token endpoint.</td>
<td>Any variable setting. For example request.queryparam.password indicates that the Password should be present as a query parameter, as, for example, ?password=changeit. To require the Password in an HTTP header, for example, set this value to request.header.password.</td>
<td>All: Optional</td>
<td></td>
</tr>
<tr>
<td>GenerateResponse</td>
<td>An element used in token endpoints, authorization endpoints, and refresh endpoints to indicate that a response should be generated for requests, and that no further processing should take place. (Indicates that the policy is an endpoint.)</td>
<td>N/A</td>
<td>All: Optional</td>
<td></td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.22.5 OAuth 2.0 Grant Types

OAuth 2.0 policies such as generate access token and generate authorization code use the GrantType method element. The below table illustrates the three supported grant types:
<table>
<thead>
<tr>
<th>Grant Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client credentials</td>
<td>&quot;Two-legged&quot; OAuth, usually implemented for trusted clients (for example, applications developed by the API provider themselves).</td>
</tr>
<tr>
<td>Authorization code</td>
<td>&quot;Three-legged&quot; OAuth, which enables the application end users to obtain an access token without exposing credentials to the application. The application requests an access token using an authorization code returned by the intermediary who authenticates the application end user. API Platform can act as both authorization server (generating authorization codes) and as a token endpoint (issuing access tokens in return for valid authorization codes).</td>
</tr>
<tr>
<td>Implicit</td>
<td>A variation on authorization code, usually enforced for browser-based applications that are implemented in scripting languages such as JavaScript</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.23 OAuth v2.0 GET

API Management generates and manages a set of OAuth resources for apps.

Depending on the OAuth configuration for an organization, API Management will generate and manage access tokens, authorization codes, and refresh tokens. For each OAuth resource that it generates, API Management also creates and stores a profile.

The GetOAuthV2Info policy type enables you to get attributes of type tokens and authorization codes and to make them available to policies and code executing in an API proxy. This policy type can be useful when you need to configure dynamic, conditional behavior based on a value in an access token.

You can attach this policy in the following locations:

```
  Code Syntax

```json
{
  "issued_at": "1847470170943",
  "application_name": "efd1903j-b667-4431-cf82-bbb3abf9t586",
  "scope": "READ",
  "status": "approved",
  "api_product_list": "[FreeProduct]",
  "expires_in": "2450",
  "developer.email": "adam@sap.com",
  "organization_id": "0",
  "refresh_token": "64XMXgDyRFpFyX0aApj1N7AGI1PnN2I2e",
  "client_id": "ceGyedEOY92OT35PemaAXYphBjCGd2N1",
  "access_token": "shTUmeI1geSK1n0TDcGLXBNe9vp",
  "organization_name": "apifactory",
  "refresh_count": "0"
}
```
The properties of an access token profile are set as variables whenever a token is generated or validated. Sometimes, however, you will need to access these properties when no token generation or validation occurs. To do so, you can explicitly populate the access token profile by using the GetOAuthV2Info policy.

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetOAuthV2Info async="false" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
   <AccessToken ref="request.access_token"></AccessToken>
   <ClientId ref="request.header.client_id"></ClientId>
</GetOAuthV2Info>
```

OAuth v2.0 GET policy defines the following elements:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessToken (Optional)</td>
<td>Use this element to retrieve the profile for an OAuth 2.0 access token.</td>
</tr>
<tr>
<td>AuthorizationCode (Optional)</td>
<td>Use this element to retrieve the profile for an OAuth 2.0 authorization code.</td>
</tr>
<tr>
<td>ClientId</td>
<td>Use this element to retrieve information about ClientId.</td>
</tr>
<tr>
<td>RefreshToken (Optional)</td>
<td>Use this element to retrieve the profile for an OAuth 2.0 refresh token.</td>
</tr>
</tbody>
</table>

OAuth v2.0 GET policy defines the following errors:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_access_token</td>
<td>The access token sent to the policy is invalid.</td>
</tr>
<tr>
<td>expired_access_token</td>
<td>The access token sent to the policy is expired.</td>
</tr>
<tr>
<td>invalid_refresh_token</td>
<td>The refresh token sent to the policy is invalid.</td>
</tr>
<tr>
<td>refresh_token_expired</td>
<td>The refresh token sent to the policy is expired.</td>
</tr>
<tr>
<td>invalid_client_expired</td>
<td>The client ID sent to the policy is invalid.</td>
</tr>
<tr>
<td>invalid_request-authorization_code_invalid</td>
<td>The authorization code sent to the policy is invalid.</td>
</tr>
<tr>
<td>authorization_code_expired</td>
<td>The authorization code sent to the policy is expired.</td>
</tr>
</tbody>
</table>

Following flow variables are populated and is used in cases where you need the profile data:
## Flow Variables

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>When</th>
<th>Variables list</th>
</tr>
</thead>
</table>
| Client ID variables   | These variables are populated when the `<ClientId>` operation executes | `oauthv2client.{policy_name}.client_id`  
`oauthv2client.{policy_name}.client_secret`  
`oauthv2client.{policy_name}.redirect_uri`  
`oauthv2client.{policy_name}.developer.email`  
`oauthv2client.{policy_name}.developer.app.name`  
`oauthv2client.{policy_name}.developer.id`  
`oauthv2client.{policy_name}.{developer_app_custom_attribute_name}` |
| Access token variables| These variables are populated when the `<AccessToken>` operation executes | `oauthv2accesstoken.{policy_name}.access_token`  
`oauthv2accesstoken.{policy_name}.scope`  
`oauthv2accesstoken.{policy_name}.refresh_token`  
`oauthv2accesstoken.{policy_name}.access_token.{custom_attribute_name}`  
`oauthv2accesstoken.{policy_name}.developer.id`  
`oauthv2accesstoken.{policy_name}.developer.app.name`  
`oauthv2accesstoken.{policy_name}.expires_in`  
`oauthv2accesstoken.{policy_name}.status` |
| Authorization code variables | These variables are populated when the `<AuthorizationCode>` operation executes | `oauthv2authcode.{policy_name}.client_id`  
`oauthv2authcode.{policy_name}.organization_id`  
`oauthv2authcode.{policy_name}.issued_at`  
`oauthv2authcode.{policy_name}.expires_in`  
`oauthv2authcode.{policy_name}.redirect_uri`  
`oauthv2authcode.{policy_name}.status`  
`oauthv2authcode.{policy_name}.state`  
`oauthv2authcode.{policy_name}.scope`  
`oauthv2authcode.{policy_name}.id`  
`oauthv2authcode.{policy_name}.{auth_code_custom_attribute_name}` |
<table>
<thead>
<tr>
<th>Variable Type</th>
<th>When</th>
<th>Variables list</th>
</tr>
</thead>
</table>
| Refresh token variables | These variables are populated when the <RefreshToken> operation executes | `oauthv2refreshtoken.{{policy_name}}.access_token`  
`oauthv2refreshtoken.{{policy_name}}.refresh_token`  
`oauthv2refreshtoken.{{policy_name}}.client_id`  
`oauthv2refreshtoken.{{policy_name}}.refresh_count`  
`oauthv2refreshtoken.{{policy_name}}.organization_name`  
`oauthv2refreshtoken.{{policy_name}}.refresh_token.expires_in`  
`oauthv2refreshtoken.{{policy_name}}.refresh_token.issued_at`  
`oauthv2refreshtoken.{{policy_name}}.refresh_token.status`  
`oauthv2refreshtoken.{{policy_name}}.developer.email`  
`oauthv2refreshtoken.{{policy_name}}.developer.id`  
`oauthv2refreshtoken.{{policy_name}}.developer.app.name`  
`oauthv2refreshtoken.{{policy_name}}.developer.app.id`  
`oauthv2refreshtoken.{{policy_name}}.access_token.{{custom_attribute_name}}` |

### 1.5.1.3.4.1.24 OAuth v2.0 SET

API Management generates and distributes OAuth access tokens to apps. API Management allows you to add or update custom attributes associated with an access token. This policy cannot be used to change fields like scope, status, expires_in, developer_email, client_id, org_name, or refresh_count. If an attribute already exists, this policy updates it. If it does not exist, the policy adds it. The access token referenced must be valid and in an approved state.

When API Management generates these OAuth artifacts, it also generates ‘profile’ that contains metadata related to the token or code. For example, the default access token profile contains name/value pairs that define expiration time, the associated app and developer, and so on.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>PreFlow</td>
<td>PostFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td></td>
</tr>
</tbody>
</table>
The JSON representation of an API Management access token looks like the following:

```json
{
    "issued_at": "1847470170943",
    "application_name": "efd1903j-b667-4431-cf82-bbb3abf9t586",
    "scope": "READ",
    "status": "approved",
    "api_product_list": "[FreeProduct]",
    "expires_in": "2450",
    "developer.email": "adam@sap.com",
    "organization_id": "0",
    "refresh_token": "64XMXgDyRFpPyX0aApjlnT7AGIPnN2Ize",
    "client_id": "ceGyedB0Y92OT35PEMaAXYphbJCGDrND",
    "access_token": "shTUmeI1geSKlnGODcGLXBe9vp",
    "organization_name": "apifactory",
    "refresh_count": "0"
}
```

In some situations, you will need to update the profile of an access token. For example, you may want to embed a tag that is unique to your business. You might need to embed a department name, a customer ID or more technically, a session identifier, in the access token.

There are two ways to do this: Using an API call or using the SetOAuthV2Info policy. You can call the management API to directly update the access token’s profile.

Use the policy when you need tokens to be updated at runtime, such as at the time when the token or code is generated by API Management.

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SetOAuthV2Info async="false" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <AccessToken ref="request.access_token"></AccessToken>
  <Attributes>
    <Attribute display="true" name="department.id">marketing</Attribute>
    <Attribute display="true" name="scope">READ, WRITE</Attribute>
  </Attributes>
</SetOAuthV2Info>
```

OAuth v2.0 SET policy defines the following elements:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessToken</td>
<td>Use the ref attribute to identify the variable where the access token is located. For example, if the access token is attached to request message as a query parameter, specify request.queryparam.access_token.</td>
</tr>
<tr>
<td>Attributes</td>
<td>A set of attributes in the access token profile that will be modified or augmented.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Attribute</td>
<td>An individual attribute to update. The name attribute identifies the property of the access token profile to be updated. For example, to modify the access token's scope property, specify scope as the value of the name attribute. The ref attribute specifies either variable or a static setting whose value will be used as the value of the access token profile property that will be updated. For example to update the attribute scope with the value READ, WRITE:</td>
</tr>
</tbody>
</table>

```sample_code```<Attribute name="scope" ref="">READ,WRITE</Attribute>```sample_code``` |

On success, the following flow variables is set:

- `oauthv2accesstoken.{policyName}.access_token`
- `oauthv2accesstoken.{policyName}.client_id`
- `oauthv2accesstoken.{policyName}.refresh_count`
- `oauthv2accesstoken.{policyName}.organization_name`
- `oauthv2accesstoken.{policyName}.expires_in`
- `oauthv2accesstoken.{policyName}.refresh_tokenExpires_in`
- `oauthv2accesstoken.{policyName}.issued_at`
- `oauthv2accesstoken.{policyName}.status`
- `oauthv2accesstoken.{policyName}.api_product_list`
- `oauthv2accesstoken.{policyName}.token_type`
- `oauthv2accesstoken.{policyName}.{custom_attribute_name}`

OAuth v2.0 SET policy defines the following errors:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_access_token</td>
<td>The access token sent to the policy is invalid.</td>
</tr>
<tr>
<td>expired_access_token</td>
<td>The access token sent to the policy is expired.</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.25 Populate Cache

An OAuth access token is written to the cache using a PopulateCache policy. The OAuth token is retrieved for subsequent requests by a LookupCache policy.

At runtime, the <PopulateCache> policy writes data from the variable you specified in the <Source> element to the cache you specified in the <CacheResource> element. You can use the <CacheKey>, <Scope>, and
<Prefix> elements to specify a key that you can use from the <LookupCache> policy to retrieve the value. Use the <ExpirySettings> element to configure when the cached value should expire.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheKey</td>
<td>Configures a unique pointer to a piece of data stored in the cache.</td>
</tr>
<tr>
<td>Prefix</td>
<td>Specifies a value to use as a cache key prefix.</td>
</tr>
<tr>
<td>KeyFragment</td>
<td>Specifies a value that should be included in the cache key, creating a namespace for matching requests to cached responses.</td>
</tr>
<tr>
<td>CacheResource</td>
<td>Specifies the cache where messages should be stored. A default cache is available.</td>
</tr>
<tr>
<td>Scope</td>
<td>Enumeration used to construct a prefix for a cache key when a &lt;Prefix&gt; element is not provided in the &lt;CacheKey&gt; element.</td>
</tr>
<tr>
<td>ExpirySettings</td>
<td>Specifies when the cached value should expire.</td>
</tr>
<tr>
<td>ExpiryDate</td>
<td>Specifies the date on which a cache entry should expire. Use the format mm-dd-yyyy.</td>
</tr>
<tr>
<td>TimeOfDay</td>
<td>The time of day at which a cache entry should expire. Use the format hh:mm:ss.</td>
</tr>
<tr>
<td>TimeoutInSec</td>
<td>The number of seconds after which a cache entry should expire.</td>
</tr>
<tr>
<td>Source</td>
<td>Specifies the variable whose value should be written to the cache.</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><!-- Token Redaction Technique 5: Grouping of Tokens
<!-- Name: Grouping of Tokens Description: Grouping of Tokens
<PopulateCache async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <CacheKey>
    <KeyFragment ref="request.header.userid"></KeyFragment>
  </CacheKey>
  <Scope>Exclusive</Scope>
  <ExpirySettings>
    <TimeoutInSec>300</TimeoutInSec>
  </ExpirySettings>
  <Source>cache-response</Source>
</PopulateCache>
```

Populate cache policy type defines the following error codes:
### Error code and Cause

<table>
<thead>
<tr>
<th>Error code</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryCannotBeCached</td>
<td>An entry cannot be cached. The message object being cached is not an instance of a class that is Serializable.</td>
</tr>
<tr>
<td>InvalidCacheResourceReference</td>
<td>The cache specified in the <code>&lt;CacheResource&gt;</code> element does not exist.</td>
</tr>
<tr>
<td>CacheNotFound</td>
<td>The cache specified in the <code>&lt;CacheResource&gt;</code> element does not exist.</td>
</tr>
</tbody>
</table>

Following fault variables is set when the policy triggers an error at runtime:

**Fault Variables**

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[prefix].[policy_name].failed</code></td>
<td><code>[prefix]: populacecache</code></td>
<td>populacecache.POP-CACHE-1.failed = true</td>
</tr>
<tr>
<td><code>[policy_name]</code></td>
<td>The name of the policy to check.</td>
<td></td>
</tr>
<tr>
<td><code>fault.[error_name]</code></td>
<td><code>error_name</code> = The specific error name to check for as listed in the table above.</td>
<td><code>fault.name</code> Matches &quot;EntryCannotBeCached&quot;</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.26 Python Script

This policy is used to configure the Python Script code to execute within the context of an API proxy.

The Python Script policy allows you to add a custom-built python functionality to your API proxy flow, wherein the functionality you need isn’t supported through the existing policies available in API Management.

Jython version 2.5.2 provides the required python language support. You can find the Jython version 2.5.2 libraries in the following link:

https://www.jython.org/jython-old-sites/docs/index.html

**Note**

The third-party libraries you add must be implemented in pure python language only.

A Python policy contains no actual code. Instead, a Python policy references a Python 'resource' and defines the Step in the API flow where the Python script executes. The Python Script resource must always have the .py extension.

You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request →</td>
<td>Response</td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
</tbody>
</table>
An example payload for the policy is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Script timeLimit="200" async="false" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <IncludeURL>py://dependency_script.py</IncludeURL>
  <ResourceURL>py://mainscript.py</ResourceURL>
</Script>
```

An example of a mainscript.py script

```
x = 1
if x == 1:
  # indented four spaces
  print "x is 1."
```

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timeLimit</code></td>
<td>Specifies the maximum time (in milliseconds) that the script is permitted to execute.</td>
<td>Yes</td>
</tr>
<tr>
<td><code>IncludeURL</code></td>
<td>This attribute specifies the python file to be loaded as dependency to the primary python file specified within the <code>ResourceURL</code> attribute. You can store the dependency python file under APIProxy/FileResources/ in the API proxy bundle. The name of the dependency python file must be of type 'String'.</td>
<td>Optional</td>
</tr>
<tr>
<td>Note</td>
<td>The python libraries you add must be implemented using pure python language only.</td>
<td></td>
</tr>
<tr>
<td><code>ResourceURL</code></td>
<td>This attribute specifies the primary python file that executes in the API flow. You must store this python file under /APIProxy/FileResources/ in the API proxy bundle. The name of the primary python file must be of type 'String'.</td>
<td>Yes</td>
</tr>
<tr>
<td>Note</td>
<td>The python libraries you add must be implemented using pure python language only.</td>
<td></td>
</tr>
</tbody>
</table>
During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidResourceUrlFormat</td>
<td>This error occurs when the format of the resource URL specified within the <code>&lt;ResourceURL&gt;</code> or the <code>&lt;IncludeURL&gt;</code> attribute of the Python Script policy is invalid, resulting in the failure of API proxy deployment.</td>
</tr>
<tr>
<td>InvalidResourceUrlReference</td>
<td>This error occurs when the <code>&lt;ResourceURL&gt;</code> or the <code>&lt;IncludeURL&gt;</code> attributes refer to a python file that doesn’t exist, resulting in failure of API proxy deployment.</td>
</tr>
<tr>
<td>NoResourceForURL</td>
<td>The <code>&lt;ResourceURL&gt;</code> and <code>&lt;IncludeURL&gt;</code> attributes refer to a Python Script file that doesn’t exist.</td>
</tr>
</tbody>
</table>

**Note**

For added security API Management python runtime executes in a restricted mode, where import is not allowed for os, sys, and java.lang.

### 1.5.1.3.4.1.27 SAML Assertion Policy

The Security Assertion Markup Language (SAML) Assertion policy enables API proxies to validate and generate SAML assertions in inbound and outbound requests, respectively.

SAML specification defines formats and protocols that enable applications to exchange XML-formatted information for authentication and authorization. A "security assertion" is a trusted token that describes an attribute of an app, an app user, or some other participant in a transaction. Security assertions are managed and consumed by two types of entities:

- **Identity providers**: Generate security assertions on behalf of participants.
- **Service providers**: Validate security assertions through trusted relationships with identity providers.

The API platform can act as an identity provider and as a service provider. It acts as an identity provider by generating assertions and attaching them to request messages, making those assertions available for processing by backend services. It acts as a service provider by validating assertions on inbound request messages.

**Generate SAML Assertion**

**Policy Processing:**

- If the message type is not XML, and IgnoreContentType is not set to true, raise a fault.
- If the Template is set, process the template as described for the AssignMessage policy. If any variables are missing and IgnoreUnresolvedVariables is not set, raise a fault.
  - If the Template is not set, construct an assertion that includes the values of the Subject and Issuer parameters or their references.
- Sign the assertion using the specified key.
- Add the assertion to the message at the specified XPath.

**Sample Schema for SAML Assertion Generation**
Sample Code

```xml
<!-- The policy will generate saml assertion and assign assertion to the
variable defined in xpath-->
<GenerateSAMLAssertion async="false" continueOnError="false" enabled="true"
ignoreContentType="false" xmlns="http://www.sap.com/apimgmt">
  <Issuer ref="saml.issuer">Issuer name</Issuer>
  <KeyStore>
    <Name >saml_key_store</Name>
    <Alias >key_store</Alias>
  </KeyStore>
  <OutputVariable>
    <FlowVariable>sapapim.assertion</FlowVariable>
    <Message name="request">
      <Namespaces>
        <Namespace prefix="env">http://schemas.xmlsoap.org/soap/envelope/</Namespace>
      </Namespaces>
      <XPath>/env:Envelope/env:Header</XPath>
    </Message>
  </OutputVariable>
  <Subject ref="saml.subject">Subject name</Subject>
  <Template ignoreUnresolvedVariables="false"><![CDATA[
    <saml2:Assertion ID="{saml.id}" IssueInstant="{saml.issueInstant}"
      Version="2.0" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <saml2:Issuer>{saml.issuer}</saml2:Issuer>
      <saml2:Subject>
        <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">{saml.subject}</saml2:NameID>
        <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
          <saml2:SubjectConfirmationData NotOnOrAfter="{sapapim.notOnorAfter}" Recipient="{saml.recipient}"/>
        </saml2:SubjectConfirmation>
        <saml2:Conditions NotBefore="{sapapim.notBefore}"
          NotOnOrAfter="{sapapim.notOnorAfter}" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
          <saml2:Audience>{saml.audience}</saml2:Audience>
        </saml2:Conditions>
      </saml2:Subject>
      <saml2:AuthnStatement AuthnInstant="{sapapim.timestamp}" SessionNotOnOrAfter="{sapapim.notOnorAfter}" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
        <saml2:AuthnContext>
          <saml2:AuthnContextClassRef urn:unchecked None"/>
        </saml2:AuthnContext>
        <saml2:AuthnContextClassRef urn:unchecked None"/>
      </saml2:AuthnStatement>
    </saml2:Assertion>]]>]]></Template>
</GenerateSAMLAssertion>
```

Element Reference
<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the policy instance. The name must be unique in the organization. Characters you can use in the name are restricted to: A-Z0-9_.-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.</td>
</tr>
<tr>
<td>ignoreContentTypetrue</td>
<td>A boolean that needs to be set to false. By default, the assertion will not be generated if the content type of the message is not an XML Content-Type. If this is set to true, then the message will be treated as XML regardless of the Content-type.</td>
</tr>
<tr>
<td>Issuer</td>
<td>The unique identifier of the identity provider. If the optional ref attribute is present, then the value of Issuer will be assigned at runtime based on the specified variable. If the optional ref attribute is not present, then the value of Issuer will be used.</td>
</tr>
<tr>
<td>KeyStore</td>
<td>The name of the KeyStore that contains the private key and the alias of the private key used to digitally sign SAML assertions.</td>
</tr>
<tr>
<td>Message</td>
<td>The target of the policy. Valid values are message, request, and response. When set to message, the policy conditionally retrieves the message object based on the attachment point of the policy. When attached to the request Flow, the policy resolves message to request, and when attached to the response Flow, the policy resolves message to response. Note that GenerateSAMLAssertion can only be attached to the TargetEndpoint request Flow. So, when using the Generate SAML Assertion policy, you should set this value to request.</td>
</tr>
<tr>
<td>XPath</td>
<td>An XPath expression that indicates the element on the outbound XML document to which the policy will attach the SAML assertion.</td>
</tr>
<tr>
<td>SignatureAlgorithm</td>
<td>SHA1 or SHA256</td>
</tr>
<tr>
<td>Subject</td>
<td>The unique identifier of the subject of the SAML assertion. If the optional ref attribute is present, then the value of Subject will be assigned at runtime based on the specified variable. If the optional ref attribute is not present, then the value of Subject will be used.</td>
</tr>
<tr>
<td>Template</td>
<td>If present, then the assertion will be generated by running this template, replacing everything denoted {} with the corresponding variable, and then digitally signing the result. The template is processed following the AssignMessage policy rules.</td>
</tr>
</tbody>
</table>

**SAML Assertion Validation**

**Policy Processing:**
The policy verifies that the inbound message request's media type is XML, by checking if the content type matches the formats text/(.*)?xml or application/(.*)?xml. If the media type is not XML, or if IgnoreContentType is not set, the policy raises a fault.

The policy parses the XML. If parsing fails, it raises a fault.

The policy validates the XML digital signature, using the values of TrustStore and ValidateSigner as described above. If validation fails, it raises a fault.

The policy checks the current timestamp (if present) against the NotBefore and NotOnOrAfter elements in the assertion.

Successful completion of the policy ensures the following:

- The digital signature on the assertion is valid and was signed by a trusted CA.
- The assertion is valid for the current time period.
- The subject and issuer of the assertion would be extracted and set in flow variables. Other policies would use these values for additional authentication, such as checking if the subject name is valid, or passing it to a target system for validation.

Sample Schema for SAML Assertion Validation

```xml
<!-- The policy will validate saml request, the saml assertion is extracted from variable defined in xpath -->
<ValidateSAMLAssertion async="false" continueOnError="false" enabled="true" ignoreContentType = "false" xmlns="http://www.sap.com/apimgmt">
  <RemoveAssertion>false</RemoveAssertion>
  <Source name="request">
    <Namespaces>
      <Namespace prefix="samlp">urn:oasis:names:tc:SAML:2.0:protocol</Namespace>
      <Namespace prefix="saml2">urn:oasis:names:tc:SAML:2.0:assertion</Namespace>
    </Namespaces>
    <XPath>/samlp:Response/saml2:Assertion</XPath>
  </Source>
  <TrustStore>saml_trust_store</TrustStore>
</ValidateSAMLAssertion>
```

### Element Reference

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name attribute</td>
<td>The name of the policy instance. The name must be unique in the organization. Characters you can use in the name are restricted to: A-Z0-9_-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.</td>
</tr>
<tr>
<td>ignoreContentType attribute</td>
<td>A boolean that needs to be set to false. By default, the assertion will not be generated if the content type of the message is not an XML Content-Type. If this is set to true then the message will be treated as XML regardless of the Content-type.</td>
</tr>
<tr>
<td>Elements and Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Source</td>
<td>The target of the policy. Valid values are message, request, and response. When set to message, the policy conditionally retrieves the message object based on the attachment point of the policy. When attached to the request Flow, the policy resolves message to request, and when attached to the response Flow, the policy resolves message to response. Note that ValidateSAMLAssertion can only be attached to the ProxyEndpoint request Flow.</td>
</tr>
<tr>
<td>XPath</td>
<td>Child of Source. An XPath expression that indicates the element on the inbound XML document from which the policy can extract the SAML assertion.</td>
</tr>
<tr>
<td>Truststore</td>
<td>The name of the TrustStore that contains trusted X.509 certificates used to validate digital signatures on SAML assertions.</td>
</tr>
<tr>
<td>RemoveAssertion</td>
<td>A boolean that can be set to true or false. When true, the SAML assertion will be stripped from the request message before the message is forwarded to the backend service.</td>
</tr>
</tbody>
</table>

The following flow variables are available after the policy is executed:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saml.id</td>
<td>The SAML assertion ID</td>
</tr>
<tr>
<td>saml.issuer</td>
<td>The &quot;Issuer&quot; of the assertion, converted from its native XML type to a string</td>
</tr>
<tr>
<td>saml.subject</td>
<td>The &quot;Subject&quot; of the assertion, converted from its native XML type to a string</td>
</tr>
<tr>
<td>saml.valid</td>
<td>Returns true or false based on the result of the validity check</td>
</tr>
<tr>
<td>saml.attribute.{attribute_name}</td>
<td>The value of the named “Attribute” present in the assertion, converted from its native XML to a string</td>
</tr>
<tr>
<td>saml.attributeNames</td>
<td>The names of all the &quot;Attributes&quot; present in the assertion, in a comma-separated list</td>
</tr>
<tr>
<td>saml.issueInstant</td>
<td>IssueInstant</td>
</tr>
<tr>
<td>saml.subjectFormat</td>
<td>Subject format</td>
</tr>
<tr>
<td>saml.scmethod</td>
<td>Subject confirmation method</td>
</tr>
<tr>
<td>saml.scdaddress</td>
<td>Subject confirmation data address</td>
</tr>
<tr>
<td>saml.scdinresponse</td>
<td>Subject confirmation data in response</td>
</tr>
<tr>
<td>saml.scdrcpt</td>
<td>Subject confirmation data recipient</td>
</tr>
</tbody>
</table>
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>saml.authnSnooa</code></td>
<td>AuthnStatement SessionNotOnOrAfter</td>
</tr>
<tr>
<td><code>saml.authnContextClassRef</code></td>
<td>AuthnStatement AuthnContextClassRef</td>
</tr>
<tr>
<td><code>saml.authnInstant</code></td>
<td>AuthnStatement AuthnInstant</td>
</tr>
<tr>
<td><code>saml.authnSessionIndex</code></td>
<td>AuthnStatement Session Index</td>
</tr>
</tbody>
</table>

#### 1.5.1.3.4.1.28 Message Validation Policy

Validates a message and rejects it if it does not conform to the specified requirements.

This policy is used to:

- Validate any XML message against an XSD schema.
- Validate SOAP messages against a WSDL definition.
- Confirm JSON or XML is well-formed, based on content-type (if `<ResourceURL>` element is omitted)

You can attach the policy in the following locations

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows

```
Sample Code

<!-- The policy will validate the response against the xsd, in the below policy
the soap message is validated to
contain a element with name "msg" and type string -->
<MessageValidation async="false" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <Element namespace="http://www.webserviceX.NET">string</Element>
  <Source>response</Source>
  <ResourceURL>xsd://validation.xsd</ResourceURL>
</MessageValidation>
```

Example validation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET">
  <xs:element name="msg" type="xsd:string"/>
</xsd:schema>
```
<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9-_-$ %. Optionally, use the &lt;DisplayName&gt; element to label the policy in the UI proxy editor with a different, natural-language name.</td>
</tr>
<tr>
<td>continueOnError</td>
<td>Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.</td>
</tr>
<tr>
<td>enabled</td>
<td>Set to true to enforce the policy. Set to false to “turn off” the policy. The policy will not be enforced even if it remains attached to a flow.</td>
</tr>
<tr>
<td>async</td>
<td>This attribute is deprecated.</td>
</tr>
<tr>
<td>DisplayName</td>
<td>Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name. If you omit this element, then the value of the policy’s name attribute is used.</td>
</tr>
<tr>
<td>Source</td>
<td>Identifies the source message to be validated. If you do not provide a &lt;Source&gt; value, a value of message is used. If the &lt;Source&gt; variable cannot be resolved, or resolves to a non-message type, then one of the following occurs: If the source variable resolves to a null value in the message flow, a steps.messagevalidation.SourceMessageNotAvailable error code is thrown. If the source variable resolves to a non-message value, a steps.messagevalidation.NonMessageVariable error code is thrown.</td>
</tr>
<tr>
<td>ResourceURL</td>
<td>Identifies the XSD schema or WSDL definition to be used to validate the source message. If the WSDL does not have schemas or if the maximum import depth exceeds 10, message validation will fail. If a &lt;ResourceURL&gt; value is not specified, the message is checked for well-formed JSON or XML if the content-type is application/json or application/xml, respectively. Default: wsdl://&lt;name&gt; Presence: Optional Type: String</td>
</tr>
<tr>
<td>SOAPMessage</td>
<td>Provides the SOAP version against which to validate SOAP messages. &lt;SOAPMessage version=&quot;1.1/1.2&quot;/&gt;</td>
</tr>
</tbody>
</table>
Elements and Attributes | Description
---|---
Version | Identifies the SOAP version against which to validate SOAP messages. Valid values: 1.1, 1.2, 1.1/1.2
Element | Specifies the root, or parent, element of the messages to be validated.


namespace | Provides the namespace of the root, or parent, element of the messages to be validated. Default: "http://sample.com"

Message Validation policy type defines the following error codes:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidResourceType</td>
<td>InvalidResourceType MessageValidation {0}: Invalid Resource Type {1}. It should be xsd or wsdl. Context {2}</td>
</tr>
<tr>
<td>ResourceCompileFailed</td>
<td>Resource CompileFailed MessageValidation {0}: Failed to compile resource {1}. Context {2}</td>
</tr>
<tr>
<td>RootElementNameUnspecified</td>
<td>RootElementNameUnspecified MessageValidation {0}: RootElement name is not specified</td>
</tr>
<tr>
<td>InvalidRootElementName</td>
<td>InvalidRootElementName MessageValidation {0}: RootElement name {1} is invalid</td>
</tr>
<tr>
<td>NonMessageVariable</td>
<td>NonMessageVariable Variable {0} does not resolve to a Message</td>
</tr>
<tr>
<td>SourceMessageNotAvailable</td>
<td>SourceMessageNotAvailable {0} message is not available for Message Validation: {1}</td>
</tr>
<tr>
<td>NoElements</td>
<td>Resource &quot;{0}&quot; has no element definitions</td>
</tr>
<tr>
<td>Failed</td>
<td>MessageValidation {0} failed with reason: &quot;{1}&quot;</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.29 Verify API Key

One of the mechanisms to prevent unauthorized access to APIs exposed over the internet is to use the verify API key policy.

The verify API key policy enforces verification of the application key in order to access your APIs.

API Management automatically generates API keys on behalf of applications. It enables API providers to view and approve API keys. By applying a policy of the type VerifyApiKey, you can enforce verification of API keys at runtime. This ensures that no application can access a protected API without a valid key.

The only setting required for the VerifyApiKey is the expected location of the API key. This policy can be attached to request or response stream of the proxy endpoint or target endpoint.
The schema for the Verify API Key policy is as follows:

**Note**
The below schema is not a working sample payload.

**Code Syntax**

```xml
<!-- The policy will prevent unauthorized users from call the api, only users with valid app key will be able to access API. The policy expects the api key to be sent as query param with name "key"-->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VerifyAPIKey async="true" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <APIKey ref="request.queryparam.key"></APIKey>
</VerifyAPIKey>
```

**Elements & Attributes**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIKey</td>
<td>The variable where the API key can be found. The API key is extracted from</td>
</tr>
<tr>
<td></td>
<td>the request message by reference to a Flow variable. For example:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;APIKey ref=&quot;request.queryparam.apikey&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td>If an application is expected to present the API key as the value of an</td>
</tr>
<tr>
<td></td>
<td>HTTP header named APIKey, then set this value to request.header.APIKey.</td>
</tr>
</tbody>
</table>

The policy populates several different types of flow variables, including:
- General
- App
- Developer
- Analytics

The following table lists the general flow variables populated by the Verify API Key policy.

These variables are all prefixed by: `verifyapikey.{policy_name}`

**For example:** `verifyapikey.{policy_name}.client_id`

**General Flow Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>The consumer key (aka API key or app key) supplied by the requesting app</td>
</tr>
<tr>
<td>client_secret</td>
<td>The consumer secret associated with the consumer key</td>
</tr>
<tr>
<td>redirection_uris</td>
<td>Any redirect URIs in the request</td>
</tr>
<tr>
<td>developer.app.id</td>
<td>The ID of the developer app making the request</td>
</tr>
</tbody>
</table>
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>developer.app.name</td>
<td>The app name of the developer app making the request</td>
</tr>
<tr>
<td>developer.id</td>
<td>The ID of the developer registered as the owner of the requesting app</td>
</tr>
<tr>
<td>DisplayName</td>
<td>The value of the policy's <code>&lt;DisplayName&gt;</code> attribute</td>
</tr>
<tr>
<td>failed</td>
<td>Set to &quot;true&quot; when API Key validation fails.</td>
</tr>
<tr>
<td>apiproduct.name*</td>
<td>The name of the API product used to validate the request</td>
</tr>
<tr>
<td>apiproduct.developer.quota.limit*</td>
<td>The quota limit set on the API product, if any</td>
</tr>
<tr>
<td>apiproduct.developer.quota.interval*</td>
<td>The quota interval set on the API product, if any</td>
</tr>
<tr>
<td>apiproduct.developer.quota.timeunit*</td>
<td>The quota time unit set on the API product, if any</td>
</tr>
</tbody>
</table>

**Note**

* API product variables are populated automatically if the API products are configured with valid environment, proxies, and resources (derived from the proxy.pathsuffix).

The following flow variables containing information about the app are populated by the policy.

These variables are all prefixed by: `verifyapikey.{policy_name}.app`.  

**Example**: `verifyapikey.{policy_name}.app.name`  

### App Flow Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the app</td>
</tr>
<tr>
<td>id</td>
<td>The ID of the app</td>
</tr>
<tr>
<td>accessType</td>
<td>Unused by API Management</td>
</tr>
<tr>
<td>callbackUrl</td>
<td>The callback URL of the app, typically used only for OAuth</td>
</tr>
<tr>
<td>DisplayName</td>
<td>The app’s display name</td>
</tr>
<tr>
<td>status</td>
<td>The app status, such as ‘approved’ or ‘revoked’</td>
</tr>
<tr>
<td>apiproducts</td>
<td>An array containing the list of API products associated with the app</td>
</tr>
<tr>
<td>appType</td>
<td>The app type is “Developer”</td>
</tr>
<tr>
<td>created_at</td>
<td>The date/time stamp when the app was created</td>
</tr>
</tbody>
</table>
Variable | Description
---|---
created_by | The e-mail address of the developer who created the app
last_modified_at | The date/time stamp when the app was last updated
last_modified_by | The e-mail address of the developer who last updated the app

The following flow variables containing information about the developer are populated by the policy. These variables are all prefixed by: `verifyapikey.{policy_name}.developer.`

For example: `verifyapikey.{policy_name}.developer.id`

**Developer Flow Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>The developer’s user name</td>
</tr>
<tr>
<td>id</td>
<td>Returns <code>org_name</code>@@<code>(developer_id)</code></td>
</tr>
<tr>
<td>firstName</td>
<td>The developer’s first name</td>
</tr>
<tr>
<td>lastName</td>
<td>The developer’s last name</td>
</tr>
<tr>
<td>e-mail</td>
<td>The developer’s e-mail address</td>
</tr>
<tr>
<td>status</td>
<td>The developer’s status, as active, inactive, or login_lock</td>
</tr>
<tr>
<td>apps</td>
<td>An array of apps associated with the developer</td>
</tr>
<tr>
<td>created_at</td>
<td>The date/time stamp when the developer was created</td>
</tr>
<tr>
<td>created_by</td>
<td>The e-mail address of the user who created the developer</td>
</tr>
<tr>
<td>last_modified_at</td>
<td>The date/time stamp when the developer was last modified</td>
</tr>
<tr>
<td>last_modified_by</td>
<td>The e-mail address of the user who modified the developer</td>
</tr>
</tbody>
</table>

The following variables are automatically populated in Analytics when a Verify API Key policy is enforced for a valid API key.

- `apiproduct.name`
- `developer.app.name`
- `client_id`
- `developer.id`

During the policy execution, the following errors can occur:
Error Code

<table>
<thead>
<tr>
<th>Error Name</th>
<th>HTTP Status</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeveloperStatusNotActive</td>
<td>401</td>
<td>The developer who created the Developer App that has the API key you are using has an inactive status. When an App Developer’s status is set to inactive, any Developer Apps created by that developer are deactivated.</td>
</tr>
<tr>
<td>FailedToResolveAPIKey</td>
<td>401</td>
<td>The policy expects to find the API key in a variable that is specified in the policy’s &lt;APIKey&gt; element. This error arises when the expected variable does not exist.</td>
</tr>
<tr>
<td>InvalidApiKey</td>
<td>401</td>
<td>An API key was received by API Management, but it is invalid. When API Management looks up the key in its database, it must exactly match the one that was sent in the request. If the API worked previously, make sure the key was not regenerated. If the key was regenerated, you will see this error if you try to use the old key.</td>
</tr>
<tr>
<td>InvalidApiKeyForGivenResource</td>
<td>401</td>
<td>An API key was received by API Management, and it is valid; however, it does not match an approved key in the Developer App associated with your API proxy through a Product.</td>
</tr>
<tr>
<td>invalid_client-app_not_approved</td>
<td>401</td>
<td>The Developer App associated with the API key is revoked.</td>
</tr>
</tbody>
</table>

Following errors can occur when you deploy a proxy containing this policy:

Deployment errors

<table>
<thead>
<tr>
<th>Error name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpecifyValueOrRefApiKey</td>
<td>The APIKey element does not have a value or key specified.</td>
</tr>
</tbody>
</table>

Following fault variables is set when the policy triggers an error at runtime:

Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prefix].[policy_name].failed</td>
<td>The fault variable [prefix] is oauthV2. The [policy_name] is the name of the policy that threw the error.</td>
<td>oauthV2.VK-VerifyAPIKey.failed = true</td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above. fault.name Matches “FailedToResolveAPIKey”</td>
<td></td>
</tr>
</tbody>
</table>

Following is an example of an error response:

```json
{
  "fault":{
    "faultstring":"Invalid ApiKey",
    "detail":{
      "errorcode":"oauth.v2.InvalidApiKey"
    }
  }
}
```
Following is an example of a fault rule:

```xml
<Sample Code>
<!-- The policy will convert xml response to json response and store in variable named jsonresponse , Attributes of a xml tag will be mapped with root "root_tag" and the attributes inside root will have names with prefix as "attributes"-->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XMLToJSON async="false" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <Options>
    <OutputSuffix>_SUFFIX</OutputSuffix>
    <OutputPrefix>PREFIX_</OutputPrefix>
    <AttributePrefix>BAR_</AttributePrefix>
    <AttributeBlockName>FOO_BLOCK</AttributeBlockName>
    <TextNodeName>TEXT</TextNodeName>
  </Options>
  <OutputPrefix>PREFIX_</OutputPrefix>
  <OutputSuffix>_SUFFIX</OutputSuffix>
</XMLToJSON>
</Sample Code>
```

### 1.5.1.3.4.1.30 XML to JSON

API Management provides policies to convert messages from the extensible markup language (XML) format to JavaScript object notation (JSON) format. This policy is called the XML to JSON.

In an ideal scenario, you often pair a JSON to XML policy on the inbound request flow with an XML to JSON policy on the outbound response flow. By combining policies this way, a JSON API can be exposed for back end services that natively support only XML.

In cases where the APIs are consumed by diverse client apps that may require either JSON or XML, you can set the response format dynamically by configuring JSON to XML and XML to JSON policies to execute with conditions.

The policy can be attached in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Preflow</code></td>
<td><code>Preflow</code></td>
</tr>
<tr>
<td><code>Flow</code></td>
<td><code>Flow</code></td>
</tr>
<tr>
<td><code>PostFlow</code></td>
<td><code>PostFlow</code></td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<Sample Code>
<Options>
  <OutputPrefix>PREFIX_</OutputPrefix>
  <OutputSuffix>_SUFFIX</OutputSuffix>
</Options>
<OutputPrefix>PREFIX_</OutputPrefix>
<AttributePrefix>BAR_</AttributePrefix>
<AttributeBlockName>FOO_BLOCK</AttributeBlockName>
AttributeBlockName>
<TextNodeName>TEXT</TextNodeName>
</Sample Code>
```
<TextAlwaysAsProperty>true</TextAlwaysAsProperty>

in conjunction if there is a namespace in the xml that is to undergo conversion -->

<!-- The below three elements have to be used in conjunction if there is a namespace in the xml that is to undergo conversion -->

<NamespaceSeparator/>
<DefaultNamespaceNodeName/>
<NamespaceBlockName/>

<NullValue>NULL</NullValue>
<RecognizeNull>true</RecognizeNull>
<RecognizeNumber>true</RecognizeNumber>
<RecognizeBoolean>true</RecognizeBoolean>

<TreatAsArray>
<Path unwrap="false">custom/xpath</Path>
</TreatAsArray>

</Options>

<!-- The variable to which the converted JSON should be assigned to -->

<OutputVariable>response</OutputVariable>

<!-- The variable that we want to convert to JSON -->

</Source>response</Source>

</XMLToJSON>

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source (optional)</td>
<td>The request or response variable that contains the XML message that you want to convert to JSON.</td>
</tr>
<tr>
<td></td>
<td>Usually, you set this to request or response, depending on whether you need to convert an inbound XML request, or an outbound XML response, into JSON.</td>
</tr>
<tr>
<td></td>
<td>If you don’t define the Source, it is treated as message. If the source variable is unresolved, or resolves to a non-message type, the policy throws an error.</td>
</tr>
<tr>
<td></td>
<td>Syntax: &lt;Source&gt;request&lt;/Source&gt;</td>
</tr>
<tr>
<td>OutputVariable (mandatory when the variable defined in the Source is a string)</td>
<td>Stores the output of the XML to JSON format conversion. This is usually the same value as the source, that is, usually inbound XML request in converted to an inbound JSON request.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify an OutputVariable, the source is treated as OutputVariable. For example, if the source is response, then OutputVariable defaults to response.</td>
</tr>
<tr>
<td></td>
<td>Syntax: &lt;OutputVariable&gt;response&lt;/OutputVariable&gt;</td>
</tr>
<tr>
<td>Options</td>
<td>Use Options to have control over the conversion from XML to JSON.</td>
</tr>
<tr>
<td></td>
<td>The following configuration elements can be added as children of the Options element. All options are optional, however, at a minimum, an empty Options element must be present for a policy to be valid.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| `<Options>`/`<RecognizeBoolean>` | Allows the conversion to maintain boolean values instead of turning them into strings.  
Valid values: true and false  
The default value is true  
If the policy configuration looks like:  

```
<RecognizeBoolean>true</RecognizeBoolean>
```

Consider an XML example:

```
<x>
  <y>false</y>
  <z>value</z>
</x>
```

If true, then:

```
{
  "x": {
    "y": false,
    "z": "value"
  }
}
```

If false, then:

```
{
  "x": {
    "y": "false",
    "z": "value"
  }
}
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Options&gt;</code>/&lt;RecognizeNumber&gt;</td>
<td>Allows the number fields in the XML payload to retain their original format if the value is true. Valid values: true or false. Default value: true.</td>
</tr>
</tbody>
</table>

If the policy configuration looks like:

```
<RecognizeNumber>true</RecognizeNumber>
```

Consider an XML example:

```
<x>
  <y>999</y>
  <z>value</z>
</x>
```

If true, then:

```
{
  "x": {
    "y": 999
    "z": "value"
  }
}
```

If false, then:

```
{
  "x": {
    "y": "999"
    "z": "value"
  }
}
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `<Options>/<RecognizeNull>`    | Allows you to convert empty values to null values. Valid values: true and false. The default value is false. If the policy configuration looks like: `<RecognizeNull>true</RecognizeNull>` Consider an XML example:  
  
  ```xml`  
  <x>  
       <y></y>  
       <z>value</z>  
  </x>  
  ```  
  If true, then:  
  ```json`  
  {  
      "x": {  
          "y": null  
      }  
  }  
  ```  
  If false, then:  
  ```json`  
  {  
      "x": {  
          "y": {},  
          "z": "value"  
      }  
  }  
  ``` |
| `<Options>/<NullValue>`        | Indicates a null value. By default the value is NULL. Syntax: `<NullValue>NULL</NullValue>` |
| `<Options>/<NamespaceSeparator>` | Use the three elements NamespaceSeparator, NamespaceBlockName, and DefaultNamespaceNodeName together. If the policy configuration looks like: `<NamespaceBlockName>#namespaceblock</NamespaceBlockName> <DefaultNamespaceNodeName>$defaultname</DefaultNamespaceNodeName> <NamespaceSeparator>:</NamespaceSeparator>` Consider the following XML example:  
  ```xml`  
  <x xmlns="http://abc.com" xmlabc:ns1="http://abc1.com">  
      <abc1:y>value</abc1:y>  
  </x>  
  ``` |
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Options&gt;/&lt;DefaultNamespaceNodeName&gt;</td>
<td>If NamespaceSeparator isn’t specified, the following JSON structure is generated:</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;x&quot;: {</td>
</tr>
<tr>
<td></td>
<td>&quot;y&quot;: &quot;value&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>If the elements NamespaceSeparator, NamespaceBlockName, and DefaultNamespaceNodeName are specified as : . #namespaceblock and $defaultname respectively, then the following JSON structure is generated:</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;x&quot;: {</td>
</tr>
<tr>
<td></td>
<td>&quot;$namespaces&quot;: {</td>
</tr>
<tr>
<td></td>
<td>&quot;%&quot;: &quot;<a href="http://abc.com">http://abc.com</a>&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;abc1&quot;: <a href="http://abc1.com">http://abc1.com</a></td>
</tr>
<tr>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td>&quot;abc1:y&quot;: &quot;value&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>&lt;Options&gt;/&lt;OutputPrefix&gt;</td>
<td>Use the two elements OutputPrefix and OutputSuffix together.</td>
</tr>
<tr>
<td></td>
<td>Syntax:</td>
</tr>
<tr>
<td></td>
<td>&lt;OutputSuffix&gt;_SUFFIX&lt;/OutputSuffix&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;OutputPrefix&gt;PREFIX_&lt;/OutputPrefix&gt;</td>
</tr>
<tr>
<td></td>
<td>Consider the following XML example: &lt;x&gt;value&lt;/x&gt;</td>
</tr>
<tr>
<td></td>
<td>If both the attributes are specified as defined in the XML to JSON example, the following JSON structure is generated:</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>PREFIX {</td>
</tr>
<tr>
<td></td>
<td>&quot;x&quot;: &quot;value&quot;</td>
</tr>
<tr>
<td></td>
<td>}_SUFFIX</td>
</tr>
<tr>
<td></td>
<td>If only OutputPrefix is specified, the following JSON structure is generated:</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>PREFIX_ {</td>
</tr>
<tr>
<td></td>
<td>&quot;x&quot;: &quot;value&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>If only OutputSuffix is specified, the following JSON structure is generated:</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;x&quot;: &quot;value&quot;</td>
</tr>
<tr>
<td></td>
<td>}_SUFFIX</td>
</tr>
<tr>
<td></td>
<td>If neither OutputPrefix nor OutputSuffix is specified, the following JSON structure is generated:</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;x&quot;: &quot;value&quot;</td>
</tr>
</tbody>
</table>
|                                      | }
Use the two attributes `AttributeBlockName` and `AttributePrefix` together, to group the values into a JSON block and append prefixes to the attribute names.

Consider the following XML example:

```xml
<a att1="value1" att2="value2"/>
```

If the policy configuration looks like:

```xml
<AttributeBlockName>FOO_BLOCK</AttributeBlockName>
<AttributePrefix>BAR_</AttributePrefix>
```

Then, the following JSON structure is generated:

```json
{
    "a": {
        "FOO_BLOCK": {
            "BAR_att1": "value1",
            "BAR_att2": "value2"
        }
    }
}
```

If only `AttributeBlockName` is specified, the following JSON structure is generated:

```json
{
    "a": {
        "FOO_BLOCK": {
            "att1": "value1",
            "att2": "value2"
        }
    }
}
```

If only `AttributePrefix` is specified, the following JSON structure is generated:

```json
{
    "a": {
        "BAR_att1": "value1",
        "BAR_att2": "value2"
    }
}
```

If neither value is specified, the following JSON structure is generated:

```json
{
    "a": {
        "att1": "value1",
        "att2": "value2"
    }
}
```

Use the two elements `TextAlwaysAsProperty` and `TextNodeName` together.

Valid values: true/false
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Options&gt;/&lt;TextNodeName&gt;</code></td>
<td>Default is false. If set to true, the content of the XML element is converted to a string property. If the policy configuration looks like:</td>
</tr>
</tbody>
</table>
|                                    | `<TextNodeName>`TEXT</TextNodeName>  
|                                    | `<TextAlwaysAsProperty>`true</TextAlwaysAsProperty>`                                                                                     |
|                                    | For the following XML:                                                                                                                      |
|                                    | `<a>`  
|                                    |    `<b>`value1</b>  
|                                    |    `<c>`value2</c>`value3</c>`value4</c>  
|                                    |    `</a>`                                                                                                                               |
|                                    | If TextAlwaysAsProperty is set to true and TextNodeName specified as TEXT, the following JSON structure is generated:                        |
|                                    | ```                                                                                                                                        |
|                                    |    {  
|                                    |        "a": {  
|                                    |            "b": {  
|                                    |                "TEXT": "value1"                                               |
|                                    |            },  
|                                    |            "c": {  
|                                    |                "TEXT": [  
|                                    |                    "value2", "value4"                                      |
|                                    |                ],  
|                                    |            "d": {  
|                                    |                "TEXT": "value3"                                           |
|                                    |            }                                                           |
|                                    |    }                                                               |
|                                    | ```  
|                                    | If TextAlwaysAsProperty is set to false and TextNodeName specified as TEXT, the following JSON structure is generated:                      |
|                                    | ```                                                                                                                                        |
|                                    |    {  
|                                    |        "a": {  
|                                    |            "b": "value1",  
|                                    |            "c": {  
|                                    |                "TEXT": [  
|                                    |                    "value2", "value4"                                      |
|                                    |                ],  
<p>|                                    |            &quot;d&quot;: &quot;value3&quot;                                                   |
|                                    |            }                                                           |
|                                    |    }                                                               |</p>
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Options&gt;</code>/<code>&lt;TreatAsArray&gt;</code></td>
<td>Enables you to streamline the values from an XML document into a JSON array. This attribute is useful when the number of child elements vary from one to many, and you want to ensure the values are always in an array.</td>
</tr>
</tbody>
</table>

Syntax:

```xml
<Options>
  <TreatAsArray>
    <Path unwrap="true">employers/employename/employees/name</Path>
  </TreatAsArray>
</Options>
```

Consider the following xml example:

**Sample Code**

```
# Example 1
<employers>
  <employename>
    <name>employer1</name>
    <employees>
      <name>emp1</name>
      <name>emp2</name>
    </employees>
  </employename>
</employers>

# Output
{
  "employers" : {
    "employename" : {
      "name" : "employer1",
      "employees" : {
        "name" : [ "emp1",
                   "emp2"
      ]
    }
  }
}
```

```
# Example 2
<employers>
  <employename>
    <name>employer1</name>
    <employees>
      <name>emp1</name>
    </employees>
  </employename>
</employers>

# Output
{
  "employers" : {
    "employename" : {
      "name" : "employer1",
      "employees" : {
        "name" : "emp1"
      }
    }
  }
}
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>By default, XML to JSON policy puts multiple child values into an array (example 1). However, when there is only one child, the policy places it in a string. In such cases <code>&lt;TreatAsArray&gt;</code> element allows you to control the output. Using the <code>&lt;TreatAsArray&gt;</code> element you can ensure that values for <code>&lt;name&gt;</code> is always put in an array, even for a single value. You configure this by identifying the Path to the element whose values you want to put in an array. Like: (consider example 2)</td>
<td></td>
</tr>
</tbody>
</table>

```xml
<TreatAsArray>
  <Path>employers/employername/employees/name</Path>
</TreatAsArray>
```

---

**Sample Code**

```
<employers>
  <employername>
    <name>employer1</name>
  </employername>
  <employees>
    <name>emp1</name>
  </employees>
</employername>
</employers>
```

# Output

```json
{
  "employers" : {
    "employername" : {
      "name" : "employer1",
      "employees" : {
        "name" : [ "emp1",
                  ]}
    }
  }
}
```

Also, in the above output, `<employername>` element and the `<name>` element for employees are unnecessary. We can unwrap them using the following syntax:

---

**Code Syntax**

```
<TreatAsArray>
  <Path>employers/employername/employees/name</Path>
  <Path>employers/employername/employees/name</Path>
</TreatAsArray>
```

**Output:**

---

**Sample Code**

```
{
  "employers" : [{
    "name" : "employer1",
```
During the policy execution, the following errors can occur:

## Error Code

<table>
<thead>
<tr>
<th>Error Name</th>
<th>HTTP Status</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>EitherOptionOrFormat</td>
<td>500</td>
<td>See the fault string.</td>
</tr>
<tr>
<td>UnknownFormat</td>
<td>500</td>
<td>See the fault string.</td>
</tr>
<tr>
<td>FormatUnavailable</td>
<td>500</td>
<td>See the fault string.</td>
</tr>
<tr>
<td>SourceUnavailable</td>
<td>500</td>
<td>The variable specified in the <code>&lt;Source&gt;</code> element has to exist.</td>
</tr>
<tr>
<td>ExecutionFailed</td>
<td>500</td>
<td>See the fault string. Be sure the incoming message contains valid XML.</td>
</tr>
<tr>
<td>InvalidSourceType</td>
<td>500</td>
<td>See the fault string.</td>
</tr>
<tr>
<td>InCompatibleTypes</td>
<td>500</td>
<td>See the fault string.</td>
</tr>
<tr>
<td>OutputVariableIsNotAvailable</td>
<td>500</td>
<td>See the fault string.</td>
</tr>
</tbody>
</table>

Following fault variables is set when the policy triggers an error at runtime:

### Fault Variables

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[prefix].[policy_name].failed</code></td>
<td>The <code>[prefix]</code> is xmltojson.. The <code>[policy_name]</code> is the name of the</td>
<td><code>xmltojson.XMLtoJSON-1.failed = true</code></td>
</tr>
<tr>
<td></td>
<td>policy that threw the error.</td>
<td></td>
</tr>
<tr>
<td><code>fault.[error_name]</code></td>
<td>[error_name] = The specific error name to check for as listed in the</td>
<td><code>fault.name Matches &quot;SourceUnavailable&quot;</code></td>
</tr>
<tr>
<td></td>
<td>table above.</td>
<td></td>
</tr>
</tbody>
</table>

Following is an example of an error response:

```
{
  "fault": {
    "faultstring": "XMLToJSON[XMLtoJSON_1]: Source xyz is not available",
    "detail": {
      "errorcode": "steps.xml2json.SourceUnavailable"
    }
  }
}
```

Following is an example of a fault rule:
**Related Information**

JSON to XML [page 170]

### 1.5.1.3.4.1.31 XSL Transform

Extensible stylesheet language transformations (XSLT) is a language that is used to convert documents from one XML format to another. This policy is used in applications that support XML but require a different XML format for the same data.

The XSL Transformation policy is executed as a processing step in an API proxy flow. The XSLT is implemented via an xsl file that is stored in the API proxy bundle under APIProxy/FileResources/<policyname>.xsl. The XSL policy references this XSL file.

The XSL policy requires two inputs:
- The name of an XSLT stylesheet (which contains a set of transformation rules) stored in the API proxy bundle under APIProxy/FileResources
- The source of the XML to be transformed (typically a request or response message)

**i Note**

<xsl:include> and <xsl:import> are not supported in the xslt code used in this policy.

You can attach this policy in the following locations:
An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XSL async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <OutputVariable>outVar</OutputVariable>
  <ResourceURL>xsl://XSLTransform.xsl</ResourceURL>
  <Source>response</Source>
</XSL>
```

Example: 
```
XSLTransform.xsl
<?xml version="1.0" encoding="UTF-8"?><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/hello-world">
  <HTML>
    <HEAD>
      <TITLE/>
    </HEAD>
    <BODY>
      <H1>
        <xsl:value-of select="greeting"/>
      </H1>
      <xsl:apply-templates select="greeter"/>
    </BODY>
  </HTML>
</xsl:template>
<xsl:template match="greeter">
  <DIV>from <I>
    <xsl:value-of select="."/>
  </I>
  </DIV>
</xsl:template>
</xsl:stylesheet>
```

### Elements and Attributes

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source (Optional)</strong></td>
</tr>
<tr>
<td>Contains the message from which information needs to be extracted. Usually this value is set to request or response, depending on whether the message to be transformed is inbound or outbound.</td>
</tr>
<tr>
<td>- If source is missing, it is treated as a simple message. For example, <code>&lt;Source&gt;message&lt;/Source&gt;</code>.</td>
</tr>
<tr>
<td>- If the source variable cannot be resolved, or resolves to a non-message type, the transformation step fails.</td>
</tr>
<tr>
<td><strong>OutputVariable (Optional)</strong></td>
</tr>
<tr>
<td>A variable that stores the output of the transformation. The OutputVariable cannot be of Message type, that is, it cannot be ‘message’, ‘request’, or ‘response’. You should set this element to be a custom variable, and then consume that variable.</td>
</tr>
<tr>
<td>To replace the message content with the output of the transformation, delete this element.</td>
</tr>
</tbody>
</table>
Elements and Attributes

<table>
<thead>
<tr>
<th>Description</th>
<th>ResourceURL (Mandatory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The XSLT file to be used for transforming the message</td>
<td></td>
</tr>
</tbody>
</table>

Parameters (Optional)

<table>
<thead>
<tr>
<th>Description</th>
<th>Parameter (Optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignores any unresolved variable errors in the XSLT script instructions. Valid values: true/false Default value: false</td>
<td></td>
</tr>
<tr>
<td>name (Mandatory)</td>
<td></td>
</tr>
<tr>
<td>Value of the parameter</td>
<td></td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLSourceMessageNotAvailable</td>
<td>{0} message is not available for XSL: {1}</td>
</tr>
<tr>
<td>XSELvaluationFailed</td>
<td>Evaluation of XSL {0} failed with reason: &quot;{1}&quot;</td>
</tr>
<tr>
<td>XSLVariableResolutionFailed</td>
<td>Failed to resolve variable {0}</td>
</tr>
<tr>
<td>XSLlInvalidResourceType</td>
<td>XSL {0}: Resource type must be xsl. Context {1}</td>
</tr>
<tr>
<td>XSLEmptyResourceUrl</td>
<td>Resource Url cannot be empty in XSL {0}</td>
</tr>
</tbody>
</table>

1.5.1.3.4.1.32  XML Threat Protection

API Management enables developers to address XML vulnerabilities and minimize attacks on API. Further, it allows you to detect XML payload attacks based on configured limits and screen against XML threats by using the following approaches:

- Validating messages against the XML schema (.xsd)
- Evaluating message content for specific blocklisted keywords or patterns
- Detecting corrupt or malformed messages before such messages are parsed

You can attach this policy in the following locations:
An example payload for the policy is as follows:

```
<XMLThreatProtection async="false" continueOnError="false" enabled="true"
 xmlns="http://www.sap.com/apimgmt">
 <NameLimits>
  <Element>20</Element>
  <Attribute>20</Attribute>
  <NamespacePrefix>20</NamespacePrefix>
  <ProcessingInstructionTarget>20</ProcessingInstructionTarget>
 </NameLimits>
 <Source>request</Source>
 <StructureLimits>
  <NodeDepth>6</NodeDepth>
  <AttributeCountPerElement>3</AttributeCountPerElement>
  <NamespaceCountPerElement>5</NamespaceCountPerElement>
  <ChildCount includeComment="true" includeElement="true" includeProcessingInstruction="true" includeText="true">5</ChildCount>
 </StructureLimits>
 <ValueLimits>
  <Text>10</Text>
  <Attribute>12</Attribute>
  <NamespaceURI>12</NamespaceURI>
  <Comment>12</Comment>
  <ProcessingInstructionData>12</ProcessingInstructionData>
 </ValueLimits>
</XMLThreatProtection>
```

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Indicates the message that needs to be screened for XML payload attacks. If it is set to request, you must validate the inbound requests from client apps. If it is set to message, the element automatically evaluates the request or response message when the message is attached to a request flow or a response flow respectively.</td>
</tr>
<tr>
<td>Source</td>
<td></td>
</tr>
</tbody>
</table>
Name Limits (Optional) | Element
--- | ---
NameLimits indicates the character limits that need to be checked and enforced by the policy.
The NameLimits <Element> element indicates the limit on the maximum number of characters allowed in an element name.
If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.

**Sample Code**

**Example**

For the following example XML:

```xml
<book category="WEB">
  <title>XML for Beginners</title>
  <author>Adam J. Smith</author>
  <year>2010</year>
</book>
```

The policy snippet below validates that the element names do not exceed the specified character limit.

```xml
<NameLimits>
  <Element>15</Element>
  <Attribute>15</Attribute>
  <NamespacePrefix>15</NamespacePrefix>
  <ProcessingInstructionTarget>15</ProcessingInstructionTarget>
</NameLimits>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Indicates a limit on the maximum number of characters allowed in an attribute name within the XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
<tr>
<td>namespacePrefix</td>
<td>Indicates a limit on the maximum number of characters allowed in the namespace prefix within the XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Example

For the following example XML:

```xml
<book category="WEB">
  <title>XML for Beginners</title>
  <author>Adam J. Smith</author>
  <year>2010</year>
</book>
```

The policy snippet below validates that the attribute name does not exceed the specified character limit.

```xml
<NameLimits>
  <Element>15</Element>
  <Attribute>15</Attribute>
  <NamespacePrefix>15</NamespacePrefix>
  <ProcessingInstructionTarget>15</ProcessingInstructionTarget>
</NameLimits>
```

For the following example XML:

```xml
<ns1:myelem xmlns:abc="http://abc.com"/>
```

The policy snippet below validates that the namespace prefix `abc` does not exceed the specified character limit.

```xml
<NameLimits>
  <Element>15</Element>
  <Attribute>15</Attribute>
  <NamespacePrefix>15</NamespacePrefix>
  <ProcessingInstructionTarget>15</ProcessingInstructionTarget>
</NameLimits>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessingInstructionTarget</td>
<td>Indicates a limit on the maximum number of characters allowed in the target of any processing instructions within the XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
<tr>
<td>StructuralLimits (Optional)</td>
<td>StructuralLimits indicates the structural limits that need to be checked and enforced by the policy. The StructuralLimits &lt;NodeDepth&gt; element indicates the maximum node depth that is allowed within an XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Attribute-CountPerElement</td>
<td>Indicates the maximum number of attributes allowed for any element within an XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
</tbody>
</table>

### Sample Code

**Example**

For the following example XML:

```xml
<book category="WEB">
  <title>XML for Beginners</title>
  <author>Adam J. Smith</author>
  <year>2010</year>
</book>
```

The policy snippet below validates that the elements book, title, author, and year do not have more than 3 attributes each. The attributes used for defining namespaces are not counted.

```xml
<StructureLimits>
  <NodeDepth>6</NodeDepth>
  <AttributeCountPerElement>3</AttributeCountPerElement>
  <NamespaceCountPerElement>5</NamespaceCountPerElement>
  <ChildCount includeComment="true" includeElement="true" includeProcessingInstruction="true" includeText="true">5</ChildCount>
</StructureLimits>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace-CountPerElement</td>
<td>Indicates the maximum number of namespace definitions allowed for any element within an XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
</tbody>
</table>

**Sample Code**

**Example**

For the following example XML:

```xml
<a1 attr1="value1" attr2="value2">
  <a2 xmlns="http://sap.com"
      xmlns:abc="http://abc.com" one="1"
      abc:two="2"/>
</a1>
```

The policy snippet below validates that for the elements `a1` and `a2`, the number of namespace definitions are limited to 5 each. In the above example, the `a1` element has 0 namespace definitions and the `a2` element has 2 namespace definitions:

```xml
<StructureLimits>
  <NodeDepth>6</NodeDepth>
  <AttributeCountPerElement>3</AttributeCountPerElement>
  <NamespaceCountPerElement>5</NamespaceCountPerElement>
  <ChildCount includeComment="true"
               includeElement="true"
               includeProcessingInstruction="true"
               includeText="true">5</ChildCount>
</StructureLimits>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildCount</td>
<td>Specifies the maximum number of child elements allowed for any element within an XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit. The ChildCount element contains the following attributes:</td>
</tr>
<tr>
<td></td>
<td>• includeComment (Default: true)</td>
</tr>
<tr>
<td></td>
<td>• includeElement (Default: true)</td>
</tr>
<tr>
<td></td>
<td>• includeProcessingInstructions (Default: true)</td>
</tr>
<tr>
<td></td>
<td>• includeText (Default: true)</td>
</tr>
</tbody>
</table>

### Sample Code

**Example**

```xml
<StructureLimits>
  <NodeDepth>6</NodeDepth>
  <AttributePerElement>3</AttributePerElement>
  <NamespacePerElement>5</NamespacePerElement>
  <ChildCount includeComment="true" includeElement="true" includeProcessingInstruction="true" includeText="true">5</ChildCount>
</StructureLimits>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValueLimits (Optional)</td>
<td>ValueLimits indicates the character limits for values to be checked and enforced by the policy. The ValueLimits &lt;Text&gt; element indicates the character limit for any text nodes within an XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
</tbody>
</table>

**Sample Code**

**Example**

For the following example XML:

```xml
<book category="WEB">
    <title>XML for Beginners</title>
    <author>Adam J. Smith</author>
    <year>2010</year>
</book>
```

The policy snippet below validates that the element text values *XML for Beginners*, *Adam J. Smith*, and *2010* do not exceed 20 characters each.

```xml
<ValueLimits>
    <Text>20</Text>
    <Attribute>10</Attribute>
    <NamespaceURI>15</NamespaceURI>
    <Comment>10</Comment>
    <ProcessingInstructionData>10</ProcessingInstructionData>
</ValueLimits>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Indicates the character limit for any attribute values within an XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
</tbody>
</table>

**Sample Code**

**Example**

For the following example XML:

```
<book category="WEB">
    <title>XML for Beginners</title>
    <author>Adam J. Smith</author>
    <year>2010</year>
</book>
```

The policy snippet below validates that the attribute value `WEB` does not exceed 10 characters.

```
<ValueLimits>
    <Text>20</Text>
    <Attribute>10</Attribute>
    <NamespaceURI>15</NamespaceURI>
    <Comment>10</Comment>
    <ProcessingInstructionData>10</ProcessingInstructionData>
</ValueLimits>
```

| NamespaceURI | Indicates the character limit for any namespace URIs within an XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit. |

**Sample Code**

**Example**

For the following example XML:

```
<ns:my_element xmlns:ns="http://ns.com"/>
```

The policy snippet below validates that the namespace URI value `http://ns.com` does not exceed 15 characters.

```
<ValueLimits>
    <Text>20</Text>
    <Attribute>10</Attribute>
    <NamespaceURI>15</NamespaceURI>
    <Comment>10</Comment>
    <ProcessingInstructionData>10</ProcessingInstructionData>
</ValueLimits>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Indicates the character limit for any comment within an XML document. If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.</td>
</tr>
</tbody>
</table>

**Sample Code**

**Example**

For the following example XML:

```xml
<book category="WEB">
  <!-- This is a comment -->
  <title>XML for Beginners</title>
  <author>Adam J. Smith</author>
  <year>2010</year>
</book>
```

The value of the `<comment>` element in the policy snippet below validates that the comment text `This is a comment` does not exceed 10 characters.

```xml
<ValueLimits>
  <Text>20</Text>
  <Attribute>10</Attribute>
  <NamespaceURI>15</NamespaceURI>
  <Comment>10</Comment>
  <ProcessingInstructionData>10</ProcessingInstructionData>
</ValueLimits>
```
## Attribute Name: ProcessingInstructionData

**Description:** Indicates the character limit for any processing instruction text within an XML document.

If you do not specify a limit, the policy applies a default value of -1, which denotes no limit.

### Sample Code

**Example**

For the following example XML:

```xml
<?xml-doc type="text/xsl" href="doc.xsl"?>
```

The policy snippet below validates that the processing instruction text `type="text/xsl" href="doc.xsl"` does not exceed 10 characters.

```xml
<ValueLimits>
  <Text>20</Text>
  <Attribute>10</Attribute>
  <NamespaceURI>15</NamespaceURI>
  <Comment>10</Comment>
  <ProcessingInstructionData>10</ProcessingInstructionData>
</ValueLimits>
```

### XML Threat Protection policy type defines the following error codes:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecutionFailed</td>
<td>Errors that occur when specific thresholds set in the policies are exceeded. These errors include <code>node depth</code>, <code>attribute count</code>, <code>child count</code>, <code>namespace count</code>, <code>element name length</code>, <code>attribute name and value length</code>, <code>namespace prefix</code> and <code>URL length</code>, <code>processing instruction name and data length</code>, <code>comment length</code>, and <code>text length</code>.</td>
</tr>
<tr>
<td>NodeDepthExceeded</td>
<td>This error occurs when the maximum depth of XML elements allowed in an XML payload is exceeded.</td>
</tr>
<tr>
<td>AttrCountExceeded</td>
<td>This error occurs when the maximum number of attributes allowed in a single element is exceeded.</td>
</tr>
<tr>
<td>ChildCountExceeded</td>
<td>This error occurs when the maximum number of child elements allowed in an XML payload is exceeded.</td>
</tr>
<tr>
<td>NSCountExceeded</td>
<td>This error occurs when the number of name spaces allowed in a single element is exceeded.</td>
</tr>
<tr>
<td>ElemNameExceeded</td>
<td>This error occurs when the maximum string length allowed in an XML tag is exceeded.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Cause</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AttrNameExceeded</td>
<td>This error occurs when the maximum length allowed for an attribute name is exceeded.</td>
</tr>
<tr>
<td>AttrValueExceeded</td>
<td>This error occurs when the maximum length allowed for an attribute value is exceeded.</td>
</tr>
<tr>
<td>NSPrefixExceeded</td>
<td>This error occurs when the namespace prefix length is exceeded.</td>
</tr>
<tr>
<td>NSURIExceeded</td>
<td>This error occurs when the namespace URL length is exceeded.</td>
</tr>
<tr>
<td>PITargetExceeded</td>
<td>This error occurs when the process instruction name length is exceeded.</td>
</tr>
<tr>
<td>PIDataExceeded</td>
<td>The allowed processing instruction data length is exceeded.</td>
</tr>
<tr>
<td>CommentExceeded</td>
<td>This error occurs when the maximum length allowed for a comment is exceeded.</td>
</tr>
<tr>
<td>TextExceeded</td>
<td>This error occurs when the maximum length allowed for text is exceeded.</td>
</tr>
<tr>
<td>SourceUnavailable</td>
<td>This error occurs when the message variable mentioned in the source element is either unavailable in the specific flow where the policy is being executed or it does not have a valid value (request, response, or message).</td>
</tr>
<tr>
<td>NonMessageVariable</td>
<td>This error occurs when the source element is set to a variable type which is not a message.</td>
</tr>
<tr>
<td>InvalidXMLPayload</td>
<td>This error occurs when the input XML Payload is invalid.</td>
</tr>
</tbody>
</table>

Following fault variables is set when the policy triggers an error at runtime:

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[prefix].[policy_name].failed</code></td>
<td>The <code>[prefix]</code> is <code>xmlattack</code>. The <code>[policy_name]</code> is the name of the policy that threw the error.</td>
<td><code>xmlattack.XPT-SecureRequest.failed = true</code></td>
</tr>
<tr>
<td>fault.[error_name]</td>
<td>[error_name] = The specific error name to check for as listed in the table above.</td>
<td><code>fault.name Matches &quot;SourceUnavailable&quot;</code></td>
</tr>
</tbody>
</table>

Following is an example of an error response:

```json
{
    "fault": {
        "faultstring": "XMLThreatProtection[XPT-SecureRequest]: Execution failed. reason: XMLThreatProtection[XPT-SecureRequest]: Exceeded object entry name length at line 2",
        "detail": {
            "errorcode": "steps.xmlthreatprotection.ExecutionFailed"
        }
    }
}
```
Following is an example of a fault rule:

```xml
<FaultRule name="XML Threat Protection Policy Faults">
  <Step>
    <Name>AM-CustomErrorResponse</Name>
    <Condition>(fault.name Matches "ExecutionFailed")</Condition>
  </Step>
  <Condition>(xmlattack.XPT-SecureRequest.failed = true)</Condition>
</FaultRule>
```

### Related Information

- JSON Threat Protection [page 177]
- Regular Expression Protection [page 296]

### 1.5.1.3.4.1.33 Regular Expression Protection

API Management helps to identify common content level threats that follow certain patterns, by enabling developers configure regular expressions that can be evaluated against API traffic at runtime.

This policy extracts information from a message (for example, URI Path, Query Param, Header, Form Param, Variable, XML Payload, or JSON Payload) and evaluates that content against predefined regular expressions. If any specified regular expressions evaluates to true, the message is considered a threat and is rejected. The most common usage of RegularExpressionProtection policy is the evaluation of payloads of JSON and XML for malicious content.

This policy supports regular expression rules as the classes in the java.util.regex package in java language.

An example payload for the policy is as follows:

```xml
<RegularExpressionProtection async="false" continueOnError="true" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>

  <!--Validates if the Uri path has "internal" keyword -->
  <URIPath>
    <Pattern>(.*)(internal)(.*)</Pattern>
  </URIPath>

  <!--Validates if  the "action" query param has any sql injection code to do any invasive operation -->
  <QueryParam name="action">
    <Pattern>\s*\((delete)|\(exec\)|(drop\s*table)|(insert)|\(shutdown\)|\(update\)|\(\bor\b\))</Pattern>
  </QueryParam>
</RegularExpressionProtection>
```
<!--Validates if the “action” header has any content with “threat” string -->
<Header name="action">
  <Pattern>(.*)\((.*)\)</Pattern>
</Header>

<!--Validates if the “action” form param has any content with “threat” string -->
<FormParam name="action">
  <Pattern>(.*)\((.*)\)</Pattern>
</FormParam>

<!--Validates if “flow.actionvar” variable has any content with “threat” string -->
<Variable name="flow.actionvar">
  <Pattern>(.*)\((.*)\)</Pattern>
</Variable>

<!--Validates if “command.action” node has any content with “threat” string -->
<JSONPayload>
  <JSONPath>
    <Expression>\$.command.action</Expression>
    <Pattern>(.*)\((.*)\)</Pattern>
  </JSONPath>
</JSONPayload>

<!--Validates if “/sap:Command/sap:Action” node has any content with “threat” string -->
<XMLPayload>
  <Namespaces>
    <Namespace prefix="sap">http://www.sap.com</Namespace>
  </Namespaces>
  <XPath>
    <Expression>/sap:Command/sap:Action</Expression>
    <Type>string</Type>
    <Pattern>(.*)\((.*)\)</Pattern>
  </XPath>
</XMLPayload>
</RegularExpressionProtection>

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
<th>Presence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Indicates the message from which information needs to be extracted.</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>If the &lt;Source&gt; element is omitted, the value defaults to message.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For example, &lt;Source&gt;message&lt;/Source&gt;. When set to message, the policy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses the request message as source when attached to a request flow.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Likewise, the policy uses the response message when attached to a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>response flow.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the source message cannot be resolved or if it resolves to a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>non-message type, the policy returns an error.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;Source&gt;response&lt;/Source&gt;</td>
<td></td>
</tr>
<tr>
<td>IgnoreUnresolvedVariables</td>
<td>If set to true and any variable is unresolvable, the policy returns an</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>error and the unresolved variable is treated as empty string (Null).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;IgnoreUnresolvedVariables&gt;false&lt;/IgnoreUnresolvedVariables&gt;</td>
<td></td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Description</td>
<td>Presence</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>URIPath</td>
<td>Specifies the request URI path from where the information needs to be extracted and evaluated against the provided regular expression. Provide at least one <code>&lt;Pattern&gt;</code> element specifying a regular expression pattern to match.</td>
<td>Optional</td>
</tr>
<tr>
<td>QueryParam</td>
<td>Specifies the request query parameter from where the information needs to be extracted and evaluated against the provided regular expression. Provide at least one <code>&lt;Pattern&gt;</code> element specifying a regular expression pattern to match.</td>
<td>Optional</td>
</tr>
<tr>
<td>Header</td>
<td>Specifies the request and response header from where the information needs to be extracted and evaluated against the provided regular expression. Provide at least one <code>&lt;Pattern&gt;</code> element specifying a regular expression pattern to match.</td>
<td>Optional</td>
</tr>
<tr>
<td>FormParam</td>
<td>Specifies the request form parameter from where the information needs to be extracted and evaluated against the provided regular expression. Provide at least one <code>&lt;Pattern&gt;</code> element specifying a regular expression pattern to match.</td>
<td>Optional</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Description</td>
<td>Presence</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>Variable</td>
<td>Specifies the variable from where the information needs to be extracted and evaluated against the provided regular expression.</td>
<td>Optional</td>
</tr>
<tr>
<td>XMLPayload</td>
<td>Specifies the XML payload from where the information needs to be extracted and evaluated against the provided regular expression.</td>
<td>Optional</td>
</tr>
<tr>
<td>JSONPayload</td>
<td>Specifies the JSON payload from where the information needs to be extracted and evaluated against the provided regular expression.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

RegularExpressionProtection policy type defines the following error codes:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>NothingToEnforce</td>
<td>RegularExpressionProtection (0): at least one of URIPath, QueryParam, Header, FormParam, XMLPayload, JSONPayload is mandatory</td>
</tr>
<tr>
<td>NoPatternsToEnforce</td>
<td>RegularExpressionProtection (0): No patterns to enforce in [1]</td>
</tr>
<tr>
<td>EmptyXPathExpression</td>
<td>RegularExpressionProtection (0): Empty XPath expression</td>
</tr>
<tr>
<td>Error code</td>
<td>Message</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EmptyJSONPathExpression</td>
<td>RegularExpressionProtection (0): Empty JSONPath expression</td>
</tr>
<tr>
<td>DuplicatePrefix</td>
<td>RegularExpressionProtection (0): Duplicate prefix (1)</td>
</tr>
<tr>
<td>NONEmptyPrefixMappedToEmptyURI</td>
<td>RegularExpressionProtection (0): Non-empty prefix (1) cannot be mapped to empty uri</td>
</tr>
<tr>
<td>ThreatDetected</td>
<td>Regular Expression Threat Detected in (0): regex: (1) input: (2)</td>
</tr>
<tr>
<td>ExecutionFailed</td>
<td>Failed to execute the RegularExpressionProtection StepDefinition (0). Reason: (1)</td>
</tr>
<tr>
<td>VariableResolutionFailed</td>
<td>Failed to resolve variable (0)</td>
</tr>
<tr>
<td>NonMessageVariable</td>
<td>Variable (0) does not resolve to a Message</td>
</tr>
<tr>
<td>SourceMessageNotAvailable</td>
<td>(0) message is not available for RegularExpressionProtection StepDefinition (1)</td>
</tr>
<tr>
<td>InvalidRegularExpression</td>
<td>RegularExpressionProtection (0): Invalid Regular Expression (1), Context (2)</td>
</tr>
<tr>
<td>InstantiationFailed</td>
<td>Failed to instantiate the RegularExpressionProtection StepDefinition (0)</td>
</tr>
<tr>
<td>CannotBe ConvertedToNodeset</td>
<td>RegularExpressionProtection (0): Result of xpath (1) cannot be converted to nodeset. Context (2)</td>
</tr>
<tr>
<td>XPathCompilationFailed</td>
<td>RegularExpressionProtection (0): Failed to compile xpath (1). Context (2)</td>
</tr>
<tr>
<td>JSONPathCompilationFailed</td>
<td>RegularExpressionProtection (0): Failed to compile jsonpath (1). Context (2)</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.34 Response Cache

This policy helps in caching data from a backend resource, thus reducing the number of requests to the resource. When applications make requests to the same URI, use this policy to return cached responses instead of forwarding all the requests to the backend server. This results in improving your API's performance through reduced latency and network traffic.

ResponseCache is useful in cases where the backend data used by your API is updated only periodically.

The maximum size for each cached object is 512 kb. You can configure the ResponseCache policy to include HTTP response headers in setting cache entry expiration and cache keys.
You can attach this policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td>Preflow</td>
<td>Preflow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ResponseCache async="true" continueOnError="false" enabled="true"
xmlns="http://www.sap.com/apimgmt">
  <CacheKey><Prefix/>
  <KeyFragment ref="request.uri" type="string"/>
  <Scope>Exclusive</Scope>
  <ExpirySettings><ExpiryDate/><TimeOfDay/>
  <TimeoutInSec ref=""/>60</TimeoutInSec>
  <SkipCacheLookup>request.header.bypass-cache = "true"</SkipCacheLookup>
  <SkipCachePopulation/>
</ResponseCache>
```

### Elements & Attributes

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DisplayName (Optional)</strong></td>
<td>Label the policy with a different (from the name attribute), natural-language name. Use this in addition to the name attribute. If you omit this element, the value of the name attribute is used. Syntax: <code>&lt;DisplayName&gt;Policy Display Name&lt;/DisplayName&gt;</code></td>
</tr>
<tr>
<td><strong>CacheLookupTimeoutInSeconds (Optional)</strong></td>
<td>This element indicates the number of seconds after which an unsuccessful cache search is considered as a missed cache. Syntax: <code>&lt;CacheLookupTimeoutInSeconds&gt;60&lt;/CacheLookupTimeoutInSeconds&gt;</code></td>
</tr>
<tr>
<td><strong>CacheResource (Optional)</strong></td>
<td>This element indicates the cache where messages are stored. To use the included shared cache, skip this element. To administratively clear entries present in the cache, specify a CacheResource by name. Syntax: <code>&lt;CacheResource&gt;my_cache_reserve&lt;/CacheResource&gt;</code></td>
</tr>
<tr>
<td>Elements &amp; Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ExcludeErrorResponse (Optional)</td>
<td>The response cache policy currently caches both success and error HTTP responses by default. The default value is <code>false</code>. If you want to exclude caching target responses with HTTP error status codes, set this element to <code>true</code>, in which case only responses with status codes from 200 to 205 (success codes) are cached. Syntax: <code>&lt;ExcludeErrorResponse&gt;true&lt;/ExcludeErrorResponse&gt;</code></td>
</tr>
<tr>
<td>SkipCacheLookup (Optional)</td>
<td>This element (if the value evaluates to <code>true</code>) indicates that cache lookup should be skipped and the cache should be refreshed. Syntax: <code>&lt;SkipCacheLookup&gt;variable-condition&lt;/SkipCacheLookup&gt;</code> In the following example, the variable for bypass-cache is set to true, indicating that in an incoming header, the cache lookup is skipped and the cache is refreshed. Example: <code>&lt;SkipCacheLookup&gt;request.header.bypass-cache = &quot;true&quot;&lt;/SkipCacheLookup&gt;</code></td>
</tr>
<tr>
<td>SkipCachePopulation (Optional)</td>
<td>This element (if the value evaluates to <code>true</code>) indicates that a write to the cache should be skipped. Syntax: <code>&lt;SkipCachePopulation&gt;variable-condition&lt;/SkipCachePopulation&gt;</code> In the following example, write to cache is skipped if the response status code is 200 or higher. Example: <code>&lt;SkipCachePopulation&gt;response.status.code &gt;= 200&lt;/SkipCachePopulation&gt;</code></td>
</tr>
</tbody>
</table>
| UseAcceptHeader (Optional)                    | This element (if set to `true`) indicates that the response cache entry’s cache key is appended with values from response accept headers. The following request headers are used while calculating the cache key:  
  - Accept  
  - Accept-Language  
  - Accept-Charset  
  - Accept-Encoding Syntax: `<UseAcceptHeader>false</UseAcceptHeader>` |
### Elements & Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **UseResponseCacheHeader**   | (Optional) This element (if set to `true`) indicates that HTTP response headers are considered while setting the time to live (TTL) of the response in the cache. While setting TTL, the values of the following response headers are considered, and compared with the values set by the **ExpireSettings** element:  
  - Cache-Control s-maxage  
  - Cache-Control max-age  
  - Expires  
  Syntax: `<UseResponseCacheHeaders>false</UseResponseCacheHeaders>` |

### Scope (Optional)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CacheKey</strong></td>
<td>(Required) The <strong>CacheKey</strong> element creates a unique pointer to a piece of data stored in cache, and has a size-limit of 2 KB each. It has two elements, KeyFragment and Prefix. The KeyFragment element indicates a value to be included in the cache key, in order to create a namespace for matching requests to cached responses. You can either provide a key (a static name) or a value (a dynamic variable) to the KeyFragment element. All the specified fragments combined (including the prefix) are concatenated to create the cache key.</td>
</tr>
<tr>
<td><strong>KeyFragment</strong></td>
<td>(Optional) <strong>KeyFragment</strong> indicates a value to be included in the cache key, in order to create a namespace for matching requests to cached responses. You can either provide a key (a static name) or a value (a dynamic variable) to the KeyFragment element. All the specified fragments combined (including the prefix) are concatenated to create the cache key.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Syntax

```xml
<KeyFragment ref="variable_name"/>
<KeyFragment>string</KeyFragment>
```

#### Example:

```xml
<KeyFragment>AccessToken</KeyFragment>
<KeyFragment ref="request_id" />
```

At runtime, the KeyFragment values are prepended with scope or prefix.

```xml
<CacheKey>
  <Prefix>User_Key</Prefix>
  <KeyFragment>AccessToken</KeyFragment>
  <KeyFragment ref="request_id" />
</CacheKey>
```
<table>
<thead>
<tr>
<th>Elements &amp; Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix (Optional)</td>
<td>The Prefix element indicates a string value that is used as a cache key prefix. Syntax: <code>&lt;Prefix&gt;prefix_string&lt;/Prefix&gt;</code> Use the prefix element along with the CacheKey and Scope elements (prefix overrides scope). If you want to specify your own value instead of a scope enumerated value, use prefix instead of scope. If you define a prefix, it prepends the cache key for entries written to the cache.</td>
</tr>
<tr>
<td>ExpirySettings (Required)</td>
<td>TimeoutInSec (Optional)</td>
</tr>
</tbody>
</table>

### Sample Code

**Example**

```xml
<CacheKey>
  <Prefix>User_Key</Prefix>
  <KeyFragment>AccessToken</KeyFragment>
  <KeyFragment ref="request_id" />
</CacheKey>
```

**Syntax**

```xml
<ExpirySettings>
  <TimeoutInSec ref="duration_variable">seconds_to_expire</TimeoutInSec>
</ExpirySettings>
```
Elements & Attributes

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOfDay</td>
</tr>
<tr>
<td>This element is a variable with the expiration time value (used in the</td>
</tr>
<tr>
<td>format \texttt{hh:mm:ss}), which indicates the time of the day when a cache</td>
</tr>
<tr>
<td>entry should expire.</td>
</tr>
<tr>
<td>The default time depends on the locale and timezone, which vary according to</td>
</tr>
<tr>
<td>where the code is running.</td>
</tr>
</tbody>
</table>

```
<ExpirySettings>
<TimeOfDay ref="time_variable">time_of_expiration</TimeOfDay>
</ExpirySettings>
```

ExpireDate
This element is a variable (used in the format \texttt{mm-dd-yyyy}) which indicates the date on which a cache entry should expire.

```
<ExpirySettings>
<ExpiryDate ref="\{date_variable\}" date_of_expiration/>
</ExpirySettings>
```

Some predefined flow variables that are populated when a ResponseCache policy is executed are described in the below table:

<table>
<thead>
<tr>
<th>Flow Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
</tr>
<tr>
<td>responsecache.{policy_name}.cachename</td>
</tr>
<tr>
<td>responsecache.{policy_name}.cachehit</td>
</tr>
<tr>
<td>responsecache.{policy_name}.invalidentry</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th>Type</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>responsecache.{policy_name}.cachename</td>
<td>String</td>
<td>Read-Only</td>
<td>This variable returns the cache used in the policy.</td>
</tr>
<tr>
<td>responsecache.{policy_name}.cachehit</td>
<td>String</td>
<td>Read-Only</td>
<td>This variable returns the cache key used in the policy.</td>
</tr>
<tr>
<td>responsecache.{policy_name}.invalidentry</td>
<td>String</td>
<td>Read-Only</td>
<td>True if the cache entry is invalid</td>
</tr>
</tbody>
</table>

Error messages that are seen when this policy triggers an error are described in the below table: responsecache.\{policy_name\}.cachekey
### Error Messages

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidTimeout</td>
<td>A negative number value was specified in the <code>&lt;CacheLookupTimeoutInSeconds&gt;</code> element.</td>
</tr>
<tr>
<td>InvalidCacheResourceReference</td>
<td>The name specified in the <code>&lt;CacheResource&gt;</code> element does not exist.</td>
</tr>
<tr>
<td>ResponseCacheStepAttachmentNotAllowedReq</td>
<td>The ResponseCache policy was attached more than once in the request path or to multiple request paths. For example, you cannot place it in both the Request PreFlow and PostFlow.</td>
</tr>
<tr>
<td>ResponseCacheStepAttachmentNotAllowedResp</td>
<td>The ResponseCache policy was attached to multiple response paths.</td>
</tr>
<tr>
<td>InvalidMessagePatternForErrorCode</td>
<td>The <code>&lt;SkipCacheLookup&gt;</code> or the <code>&lt;SkipCachePopulation&gt;</code> element in a ResponseCache policy contained an invalid condition.</td>
</tr>
<tr>
<td>CannotDeleteStepDefinition</td>
<td>You must detach the policy definition from the proxy flows before you can delete the policy.</td>
</tr>
<tr>
<td>CacheNotFound</td>
<td>The cache specified in the <code>&lt;CacheResource&gt;</code> element does not exist.</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.1.35 Statistics Collector Policy

This policy enables you to collect statistics for data in a message, such as product ID, price, REST action, client and target URL, and message length.

The data comes from flow variables predefined by API Management or custom variables that you define. The statistics data is passed to the analytics server, which analyzes the statistics and generates reports. This can be viewed by creating custom charts.

Only one Statistics Collector policy should be attached to a single API proxy. If there are multiple Statistics Collector policies in a proxy, then the last one to execute determines the data written to the analytics server.

You can attach the policy in one of the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Response</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>PreFlow</td>
<td>PreFlow</td>
</tr>
<tr>
<td>Flow</td>
<td>Flow</td>
</tr>
<tr>
<td>PostFlow</td>
<td>PostFlow</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An example payload for the policy is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<StatisticsCollector xmlns="http://www.sap.com/apimgmt">
  <Statistics>
    <Statistic name="productID" ref="product.id" type="string">999999</Statistic>
    <!-- The policy collects data for each request and passes to the analytics server, in the below policy, the data is collected from productID and the price variables. if the variables are not set, the default values are used -->
</StatisticsCollector>
```
<Statistic name="price" ref="product.price" type="string">0</Statistic>
</Statistics>
</StatisticsCollector>

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name used to reference the data collected for the specified variable. When viewing analytics data, use this name to reference the data collected about the variable specified by the ref attribute. If the variable specified by ref is undefined on a request or response, then default-StatValue specifies the value collected for the variable. If you omit the default value, no data is collected for the variable when the variable is undefined. The following restrictions apply:</td>
</tr>
<tr>
<td></td>
<td>- Names cannot be multiple-word (no spaces).</td>
</tr>
<tr>
<td></td>
<td>- No spaces, m dashes, quotation marks, underscores, hyphens, or periods.</td>
</tr>
<tr>
<td>ref</td>
<td>The flow variable for which you are collecting statistics. This variable can be a flow variable predefined by API Management or a custom variable that you define in your API proxy. The ref attribute often references a custom variable defined by the Extract Variables policy.</td>
</tr>
<tr>
<td>type</td>
<td>Specifies the data type of the variable specified by the ref attribute. Valid values are: string/integer/float/long/double. For data of type string, reference the statistical data as a Dimension in a custom report. For numerical data types (integer/float/long/double), reference the statistical data in a custom report as a Metric. See Create custom reports for more. The value of type can be omitted only if ref refers to a predefined API Management flow variable or the type is declared in the XML payload of the Extract Variables policy.</td>
</tr>
<tr>
<td>Default Value</td>
<td>If you omit this element, the value of the policy’s name attribute is used.</td>
</tr>
</tbody>
</table>

During the policy execution, the following errors can occur:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Fault String</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidName</td>
<td>StatisticsCollection: Name: [name] conflicts with system defined variables. Context [context]</td>
<td>See the fault string.</td>
</tr>
<tr>
<td>DatatypeMissing</td>
<td>StatisticsCollection [name]: Datatype of [type] is missing. Context [context]</td>
<td>See the fault string.</td>
</tr>
</tbody>
</table>
1.5.1.3.4.1.36  Open Connectors

API management provides open connector policy to be attached to an Open Connector type API.

For an open connector type API, you can attach only one open connector policy. The policy is either attached to the target endpoint or the proxy endpoint.

You can attach the policy in the following locations:

<table>
<thead>
<tr>
<th>ProxyEndpoint</th>
<th>TargetEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Preflow</td>
</tr>
<tr>
<td></td>
<td>•</td>
</tr>
<tr>
<td>PostFlow</td>
<td>Flow</td>
</tr>
</tbody>
</table>

An example payload of the payload is as follows:

```xml
<OpenConnectors async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <InstanceSecret kvm-map-name="apim.oc.instance.token" kvm-key-name="default"/>
</OpenConnectors>
```

For API proxies that are created via service or imported, before attaching the policy, ensure that there is a KVM created with map name same as 'kvm-map-name' and KVM key with 'kvm-key-name'.

Following table lists the elements and attributes that you can configure on this policy.

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceSecret</td>
<td>Specifies the KVM map name and key.</td>
</tr>
<tr>
<td>kvm-map-name</td>
<td>Map name that specifies the instance token.</td>
</tr>
<tr>
<td>kvm-key-name</td>
<td>Specifies the key to the KVM.</td>
</tr>
</tbody>
</table>

Open Connector Callout

This section describes the process to add a service callout policy to an open connector type API.

In the policy editor window, choose open connector policy. In the create policy window, enter the required details. Select the External Call checkbox. From the API Provider dropdown, discover an API Provider (Only open connector type Providers are listed here)).

An example payload of the payload is as follows:

```xml
<OpenConnectors async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <InstanceSecret kvm-map-name="apim.oc.instance.token" kvm-key-name="default"/>
</OpenConnectors>
```

# For API proxies that are created via service or imported, before attaching the policy, ensure that there is a KVM created with map name same as 'kvm-map-name' and KVM key with 'kvm-key-name'.

SAP API Management in the Cloud Foundry Environment
<!-- Environment and Proxy scoped variables will be retrieved and passed as authorization header to Backend -->
<OpenConnectorCallout xmlns="http://www.sap.com/apimgmt" async="true" continueOnError="false" enabled="true">
  <Request>
    <Set>
      <Headers/>
      <FormParams/>
      <QueryParams/>
      <Path>/elements/api-v2/ping</Path>
      <Verb>GET</Verb>
      <Payload/>
    </Set>
  </Request>
  <APIProvider>OcSanity</APIProvider>
  <Timeout>30000</Timeout>

<!-- PUT stores the key value pair mentioned inside the element -->
<InstanceSecret kvm-map-name="apim.occallout.instance.secret.Test5512" kvm-key-name="Haha_MyGmail"/>
</OpenConnectorCallout>

# For API proxies that are created via service or imported, before attaching the policy, ensure that there is a KVM created with map name same as 'kvm-map-name' and KVM key with 'kvm-key-name'.

Following table lists the elements and attributes that you can configure on this policy.

<table>
<thead>
<tr>
<th>Elements and Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APProvider</td>
<td>Name of the open connector type API Provider.</td>
</tr>
<tr>
<td>Timeout</td>
<td>Time the policy waits for a response from the target. Time in milliseconds</td>
</tr>
<tr>
<td>Headers</td>
<td>Overwriting existing HTTP headers.</td>
</tr>
<tr>
<td>QueryParams</td>
<td>Adds new query parameters to the request. Use this attribute for GET operation only.</td>
</tr>
</tbody>
</table>

### Headers

** Hóa Sample Code**

This example sets the 'test' header to `request.header.test`.

```xml
<Headers>
  <Header name="test">{request.header.test}</Header>
</Headers>
```

### QueryParams

** Hóa Sample Code**

The following example sets the "address" query parameter to the value of the request.header.address variable:

```xml
<QueryParams>
  <QueryParam name="address">{request.header.address}</QueryParam>
</QueryParams>
```
### Elements and Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceSecret</td>
<td>Specifies the KVM map name and key.</td>
</tr>
<tr>
<td>kvm-map-name</td>
<td>Map name that specifies the instance token.</td>
</tr>
<tr>
<td>kvm-key-name</td>
<td>Specifies the key to the KVM.</td>
</tr>
</tbody>
</table>

#### 1.5.1.3.4.2 Variable References

API Management describes a set of variables predefined by the API Services.

Types of variables available are:

- Request Message Variables [page 324]
- System Variables [page 337]
- Response Message Variables [page 345]
- Message Variables [page 343]
- Error Variables [page 340]
- Configuration Variables [page 339]
- Path Variables [page 349]
- API Proxy Flows Variables [page 342]
- TLS/SSL Connection Information Variables [page 340]
### 1.5.1.3.4.2.1 Flow Variables

This topic provides information about the flow variables.

**Client Flow Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope begins</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>client.cn</td>
<td>The common name specified in the TLS/SSL certificate presented by the client app.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
</tbody>
</table>

**Note**

If the common name in the TLS/SSL certificate contains comma separated values, then the client.cn flow variable returns only the first value before comma. If you need to read all the values, then it is recommended to use the tls.client.s.dn flow variable. For more information, see [TLS/SSL Connection Information Variables](#page 340).

This behaviour applies to all the client flow variables that are associated with TLS/SSL context. For example, client.country, client.email.address, client.organization, client.organization.unit, client...
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope begins</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>client.country</td>
<td>The country/region in the TLS/SSL certificate presented by the client app.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.email.address</td>
<td>The e-mail address in the TLS/SSL certificate presented by the client app.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.host</td>
<td>The HTTP host IP associated with the request received by the ProxyEndpoint.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.ip</td>
<td>The IP address of the client or system sending the message. For example, this could be the original client IP or a load balancer IP.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.locality</td>
<td>The locality (City) in the TLS/SSL certificate presented by the client.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.organization</td>
<td>The organization in the TLS/SSL certificate presented by the client.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.organization.unit</td>
<td>The organizational unit in the TLS/SSL certificate presented by the client.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.port</td>
<td>The HTTP port associated with the originating client request to ProxyEndpoint.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope begins</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| client.received.end.time       | - The time, expressed in string form, at which the proxy finished receiving the request from the originating client at the ProxyEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC  
  - This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413. | Proxy request  | String  | Read       |
<p>| client.received.end.timestamp  | The timestamp value specifying when the proxy finished receiving the request from the originating client at the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC. | Proxy request  | Long    | Read       |</p>
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope begins</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>client.received.start.time</td>
<td>• The time, expressed in string form, at which the proxy began receiving the request from the originating client at the ProxyEndpoint. For example: ‘Wed, 21 Aug 2013 19:16:47 UTC’&lt;br&gt;• This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, ‘Wed, 21 Aug 2013 19:16:47 UTC’ corresponds to the timestamp value of 1377112607413.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.received.start.timestamp</td>
<td>The timestamp value specifying when the proxy began receiving the request from the originating client at the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Proxy request</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope begins</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
<td>--------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| client.sent.end.time | ● The time, expressed in string form, when the ProxyEndpoint finished returning the response to the originating client app. For example: Wed, 21 Aug 2013 19:16:47 UTC<sup>1</sup>  
  ● This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413. | Proxy response        | String | Read       |
<p>| client.sent.end.time-stamp | The timestamp value specifying when the ProxyEndpoint finished returning the response to the originating client app. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC. | Proxy response        | Long   | Read       |</p>
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope begins</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
</table>
| client.sent.start.time  | • The time, expressed in string form, when the ProxyEndpoint started returning the response to the originating client app. For example: Wed, 21 Aug 2013 19:16:47 UTC  
● This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, ‘Wed, 21 Aug 2013 19:16:47 UTC’ corresponds to the timestamp value of 1377112607413. | Proxy response | String | Read       |
| client.sent.start.time-stamp | The timestamp value specifying when the ProxyEndpoint started returning the response to the originating client app. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC. | Proxy response | Long   | Read       |
| client.scheme           | Returns http or https depending on the transport used by client app to send the request message.                                                                                                            | Proxy request  | String | Read       |
| client.state            | The state in the TLS/SSL certificate presented by the client.                                                                                                                                               | Proxy request  | String | Read       |
| client.ssl.enabled      | Returns true or false, depending on whether the ProxyEndpoint is configured for TLS/SSL.                                                                                                                                 | Proxy request  | String | Read       |
## Error Flow Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>Error of type Message, which is a contextual object in the error flow</td>
<td>Error</td>
<td>Message</td>
<td>Read/Write</td>
</tr>
<tr>
<td>error.content</td>
<td>Content of the error</td>
<td>Error</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>error.message</td>
<td>Message associated with an error, whose value is available only before the error Flow is executed.</td>
<td>Error</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>error.status.code</td>
<td>The HTTP status code associated with the error. For example: &quot;400&quot;.</td>
<td>Error</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>error.reason.phrase</td>
<td>The reason phrase associated with the error. For example: &quot;Bad Request&quot;</td>
<td>Error</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>error.transport.message</td>
<td>Any error of type TransportMessage</td>
<td>Error</td>
<td>Transport_Message</td>
<td>Read</td>
</tr>
<tr>
<td>error.state</td>
<td>State in the Flow where an error occurred</td>
<td>Error</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>error.header.&lt;name&gt;</td>
<td>Get or set the response header.</td>
<td>Error</td>
<td>integer</td>
<td>Read/Write</td>
</tr>
</tbody>
</table>

## Request Flow Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>The complete request, including any payload present.</td>
<td>Proxy request</td>
<td>Message</td>
<td>Read</td>
</tr>
<tr>
<td>request.content</td>
<td>Gets or sets the payload of the request message.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>request.formparam.{formparam_name}</td>
<td>Gets or sets the value of the specified form parameter in the request sent from the client app.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>request.formparam.{formparam_name}.values.count</td>
<td>Count of all values for the specified form parameter associated with the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.formparams.count</td>
<td>Count of all form parameters associated with the request sent from the client app.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>request.formparams.names</td>
<td>A list of all form parameter names associated with the request.</td>
<td>Proxy request</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>request.header.{header_name}</td>
<td>Gets or sets the value of a particular header found in the request.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the {header_name} has multiple values, you can read or retrieve all the values of the header. For more information on extracting multiple values of an HTTP header, see Multi-value HTTP Headers in an API Proxy [page 350]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>request.header.{header_name}.values</td>
<td>All the values of a particular header in the request</td>
<td>Proxy request</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>request.header.{header_name}.values.count</td>
<td>Count of all the values of a particular header in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.formparams.count</td>
<td>Count of all form parameters associated with the request sent from the client</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.formparams.names</td>
<td>A list of all form parameter names associated with the request.</td>
<td>Proxy request</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>request.header.{header_name}</td>
<td>Gets or sets the value of a particular header found in the request.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>request.header.{header_name}.values</td>
<td>All the values of a particular header in the request.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>request.header.{header_name}.values.count</td>
<td>Count of all the values of a particular header in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.headers.count</td>
<td>Count of all the headers in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>request.path</td>
<td>The unproxied resource path (not including the host) to the backend service, excluding query parameters.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.queryparam.{queryparam_name}</td>
<td>The value of a particular query parameter found in the request.</td>
<td>Proxy request</td>
<td>string</td>
<td>Read/Write</td>
</tr>
<tr>
<td>request.queryparam.{queryparam_name}.values.count</td>
<td>The count of all the values of a particular query parameter in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.queryparams.count</td>
<td>The count of all the query parameters in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.queryparams.names</td>
<td>request.queryparams.names</td>
<td>Proxy request (Java Object)</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>request.querystring</td>
<td>The complete list of query parameters in the request sent from the client app. For example, if the request is <a href="http://host.com/123?name=first&amp;surname=second&amp;place=address">http://host.com/123?name=first&amp;surname=second&amp;place=address</a> then this variable returns name=first&amp;surname=second&amp;place=address</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>request.transportid</td>
<td>ID of the request as type TransportMessage which is a contextual object</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>request.transport.message</td>
<td>Request of type TransportMessage which is a contextual object</td>
<td>Proxy request</td>
<td>Transport</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>request.uri</td>
<td>In an API proxy, the proxy <code>&lt;BasePath&gt;</code> in the ProxyEndpoint (in addition to the proxy’s base URL) maps to the target service URL in the TargetEndpoint. For example: <code>&lt;ProxyEndpoint&gt;</code> ... <code>&lt;BasePath&gt;/my-mock-proxy&lt;/BasePath&gt;</code> points to <code>&lt;TargetEndpoint&gt;</code> ... <code>&lt;HTTPTargetConnection&gt; http://mocktarget.sap.com &lt;/HTTPTargetConnection&gt;</code></td>
<td>Proxy request (differs with response)</td>
<td>String</td>
<td>Read</td>
</tr>
</tbody>
</table>

In the request, the request.uri is the proxy base path + the remainder of the address, including query parameters.

In the response, the request.uri is the remainder of the address, including query parameters, after the HTTPTargetConnection.

The difference is because the original request came into the proxy, but then the proxy makes another...
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>request to the target service. Let’s say the following call is made to our sample proxy, which has a base path of /my-mock-proxy: <a href="http://my_org-test.sap.com/my-mock-proxy/user?user=test">http://my_org-test.sap.com/my-mock-proxy/user?user=test</a> and the proxy calls <a href="http://mocktarget.get.apigee.net">http://mocktarget.get.apigee.net</a> (which appends /user?user=test to that URL). Request: request.uri = /my-mock-proxy/user?user=test Response: request.uri = /user?user=test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>request.url</td>
<td>Returns the exact complete URL of the final request made.</td>
<td>Target request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>route.name</td>
<td>The name of the RouteRule that was executed in the ProxyEndpoint. For example: default. A RouteRule references an API proxy TargetEndpoint to execute.</td>
<td>Target request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>route.target</td>
<td>The name of the TargetEndpoint that was executed. For example: default.</td>
<td>Target request</td>
<td>String</td>
<td>Read</td>
</tr>
</tbody>
</table>

**Response Flow Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope begins</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>response</td>
<td>Complete response message returned by target</td>
<td>Target response</td>
<td>Message</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.content</td>
<td>Payload content of the response message returned by the target</td>
<td>Target response</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.formparam.</td>
<td>The value of a form parameter in the response</td>
<td>Target response</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>{formparam_name}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>response.formparam.</td>
<td>Count of all the values of the specified form parameter in response</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>{formparam_name}.values.count</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>response.formparams.count</td>
<td>Count of all form parameters in the response</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>response.formparams.names</td>
<td>The names of all the form parameters in the response</td>
<td>Target response</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope begins</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>-----------</td>
<td>------------------</td>
</tr>
<tr>
<td>response.header.{header_name}</td>
<td>Gets or sets the value of a specified HTTP header in the response. If the header has multiple values (such as a CSV list), a GET returns the first value only.</td>
<td>Target response</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.header.{header_name}.values</td>
<td>All the values of a specified HTTP header in response.</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>response.headers.count</td>
<td>Count of all the headers in the response</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>response.headers.names</td>
<td>The names of all the headers in the response</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>response.reason.phrase</td>
<td>The response reason phrase for a particular request</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>response.status.code</td>
<td>The response code returned for a request. You can use this variable to override the response status code, which is stored in message.status.code.</td>
<td>Target response</td>
<td>Integer</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.transport.message</td>
<td>Response of type TransportMessage which is a contextual object</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
</tbody>
</table>
1.5.1.3.4.2.2 Request Message Variables

Request message variables are used in policies to access message components like the header, the query parameters, form parameters, the source IP address, the HTTP message body.

API proxy applies the incoming request to a series of policies, depending on the request condition, API proxy can either modify or transform the request. Based on the content of the request variable, policies can either transform or reject the request. Supported request variables are listed in the Request Message Variables table.

Request Message Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>client.host</td>
<td>The HTTP host IP associated with the request received by the ProxyEndpoint.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.ip</td>
<td>The IP address of the client or system sending the message to the Edge router. For example, this could be the original client IP or a load balancer IP.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.port</td>
<td>The HTTP port associated with the originating client request to ProxyEndpoint.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>client.received.end.timestamp</td>
<td>The timestamp value specifying when the proxy finished receiving the request from the originating client at the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Proxy request</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>client.received.start.time</td>
<td>The time, expressed in string form, at which the proxy began receiving the request from the originating client at the ProxyEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC. This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.received.start.timestamp</td>
<td>The timestamp value specifying when the proxy began receiving the request from the originating client at the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Proxy request</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>client.sent.end.time</td>
<td>The time, expressed in string form, when the ProxyEndpoint finished returning the response to the originating client app. For example: Wed, 21 Aug 2013 19:16:47 UTC. This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.sent.end.timestamp</td>
<td>The timestamp value specifying when the ProxyEndpoint finished returning the response to the originating client app. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Proxy request</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>client.sent.start.time</td>
<td>The time, expressed in string form, when the ProxyEndpoint started returning the response to the originating client app. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.sent.start.timestamp</td>
<td>The timestamp value specifying when the ProxyEndpoint started returning the response to the originating client app. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Proxy request</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>client.scheme</td>
<td>Returns http or https depending on the transport used by client app to send the request message.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>message.queryparam.</td>
<td>Returns the specified message query parameter.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>{queryparam_name}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>message.queryparam.</td>
<td>The total count of a specified query parameter associated with the request sent to the ProxyEndpoint from the client app</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>{queryparam_name}.values.count</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>---------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>message.queryparams.count</td>
<td>The total count of all query parameters associated with the request sent to the ProxyEndpoint from the client app.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>message.queryparams.names</td>
<td>A list of all query parameter names associated with the request sent to the ProxyEndpoint from the client app.</td>
<td>Proxy request</td>
<td>Collection (Java Object)</td>
<td>Read</td>
</tr>
<tr>
<td>message.querystring</td>
<td>A string containing all query parameter names and values associated with the request sent to the ProxyEndpoint from the client app.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>message.uri</td>
<td>The complete URI path (following the domain URL) including query parameters.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>message.verb</td>
<td>The HTTP verb (GET, PUT, POST, DELETE, and so on) associated with the request</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>message.version</td>
<td>The HTTP version associated with the request sent to the ProxyEndpoint from the client app.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>messageid</td>
<td>This ID is logged in the error logs to correlate the messageid with the errors.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>proxy.basepath</td>
<td>The value of the Base Path in your API proxy configuration. The base path is the URI fragment that follows the host in the URL. Conditional flow URIs follow the base path.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>proxy.client.ip</td>
<td>The <strong>X-Forwarded-For</strong> address of the inbound call, which is the IP address API Management received from the last external TCP handshake. This could be the calling client.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>proxy.pathsuffix</td>
<td>The value of API proxy basepath suffix that is sent from the client and received at the ProxyEndpoint. The basepath is defined as the path component that uniquely identifies the API proxy. The public-facing URL of an API proxy is comprised of your organization name, the environment where the proxy is deployed, the basepath, the basepath suffix, and any query parameters. For example, in the following request: <a href="http://myorg-test.sap.net/v2/weatherapi/forecastss?w=12797282">http://myorg-test.sap.net/v2/weatherapi/forecastss?w=12797282</a> The basepath suffix is /forecastss</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>proxy.url</td>
<td>Gets the complete URL associated with the proxy request received by the ProxyEndpoint, including any query parameters present. Note that the host in proxy.url is the router host, not the host used in the original request.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>request</td>
<td>The complete request, including any payload present.</td>
<td>Proxy request</td>
<td>Message</td>
<td>Read</td>
</tr>
<tr>
<td>request.content</td>
<td>Gets or sets the payload of the request message.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>request.formparam. {formparam_name}</td>
<td>Gets or sets the value of the specified form parameter in the request sent from the client app.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>request.formparam. {formparam_name}.values.count</td>
<td>Count of all values for the specified form parameter associated with the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.formparams.count</td>
<td>Count of all form parameters associated with the request sent from the client app.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.formparams.names</td>
<td>A list of all form parameter names associated with the request.</td>
<td>Proxy request</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>request.header. {header_name}</td>
<td>Gets or sets the value of a particular header found in the request.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>request.header. {header_name}.values</td>
<td>All the values of a particular header in the request.</td>
<td>Proxy request</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>request.header. {header_name}.values.count</td>
<td>Count of all the values of a particular header in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.headers.count</td>
<td>Count of all the headers in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.path</td>
<td>The un-proxied resource path (not including the host) to the backend service, excluding query parameters.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>request.queryparam. {queryparam_name}</td>
<td>The value of a particular query parameter found in the request.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>request.queryparam. {queryparam_name}.values.count</td>
<td>The count of all the values of a particular query parameter in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>-----------------------</td>
<td>------------</td>
</tr>
<tr>
<td>request.queryparams.count</td>
<td>The count of all the query parameters in the request.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>request.queryparams.names</td>
<td>The names of all the query parameters in the request.</td>
<td>Proxy request</td>
<td>Collection (JavaObject)</td>
<td>Read</td>
</tr>
<tr>
<td>request.querystring</td>
<td>The complete list of query parameters in the request sent from the client app.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>For example, if the request is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="http://host.com/123?name=first&amp;surname=second&amp;place=address">http://host.com/123?name=first&amp;surname=second&amp;place=address</a></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>then this variable returns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>name=first&amp;surname=second&amp;place=address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>request.transportid</td>
<td>ID of the request as type TransportMessage which is a contextual object</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>request.transport.message</td>
<td>Request of type TransportMessage which is a contextual object</td>
<td>Proxy request</td>
<td>Transport message</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>request.uri</td>
<td>In an API proxy, the proxy <code>&lt;BasePath&gt;</code> in the ProxyEndpoint (in addition to the proxy’s base URL) maps to the target service URL in the TargetEndpoint. For example: &lt;ProxyEndpoint&gt; ...&lt;BasePath&gt;/my-mock-proxy&lt;/BasePath&gt; points to &lt;TargetEndpoint&gt; ...&lt;HTTPTargetConnection&gt; <a href="http://mocktarget.sap.com">http://mocktarget.sap.com</a> &lt;/HTTPTargetConnection&gt; In the request, the request.uri is the proxy base path + the remainder of the address, including query parameters. In the response, the request.uri is the remainder of the address, including query parameters, after the HTTPTargetConnection. The difference is because the original request came into the proxy, but then the proxy makes another</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>request to the target service.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Let’s say the following call is made to our sample proxy, which has a base path of /my-mock-proxy:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="http://my_org-test.sap.com/my-mock-proxy/user">http://my_org-test.sap.com/my-mock-proxy/user</a>? user=test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>and the proxy calls</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="http://mocktarget.get.apigee.net">http://mocktarget.get.apigee.net</a> (which appends /user? user=test to that URL).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Request: request.uri = /my-mock-proxy/user? user=test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: request.uri = /user?user=test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>request.url</td>
<td>Returns the exact complete URL of the final request made.</td>
<td>Target request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>request.verb</td>
<td>The HTTP verb used for the request. For example: GET, PUT, DELETE</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>request.version</td>
<td>Gets the HTTP version of the request.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>response.formstring</td>
<td>The complete list of form parameters in the request.</td>
<td>Target request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>For example: name=test&amp;type=first &amp;group=A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>route.name</td>
<td>The name of the RouteRule that was executed in the ProxyEndpoint. For example: default. A RouteRule references an API proxy TargetEndpoint to execute.</td>
<td>Target request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>route.target</td>
<td>The name of the TargetEndpoint that was executed. For example: default.</td>
<td>Target request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>servicecallout.requesturi</td>
<td>The TargetEndpoint URI for a ServiceCallout policy. The URI is the TargetEndpoint URL without the protocol and domain specification.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>servicecallout.{policy-name}.expectedcn</td>
<td>The expected Common Name of the TargetEndpoint as referred to in a ServiceCallout policy. This is meaningful only when the TargetEndpoint refers to an TLS/SSL endpoint.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>servicecallout.{policy-name}.target.url</td>
<td>The TargetEndpoint URL for a particular ServiceCallout policy.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>target.basepath</td>
<td>Returns basepath of TargetEndpoint</td>
<td>Target Request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.copy.pathsuffix</td>
<td>When true, request forwarded from ProxyEndpoint to TargetEndpoint retains path suffix (the URI path fragment following the URI defined in the ProxyEndpoint base path.)</td>
<td>Target Request</td>
<td>Boolean</td>
<td>Read/Write</td>
</tr>
<tr>
<td>target.copy.queryparams</td>
<td>When true, request forwarded from ProxyEndpoint to TargetEndpoint retains query parameters.</td>
<td>Target Request</td>
<td>Boolean</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>target.cn</td>
<td>The Common Name of the TargetEndpoint. This is meaningful only when the TargetEndpoint refers to an TLS/SSL endpoint.</td>
<td>Target request</td>
<td>Boolean</td>
<td>Read</td>
</tr>
<tr>
<td>target.expectedcn</td>
<td>The expected Common Name of the TargetEndpoint. This is meaningful only when the TargetEndpoint refers to an TLS/SSL endpoint.</td>
<td>Proxy Request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.name</td>
<td>Target to which message is reaching from targetendpoint</td>
<td>Target Request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.sent.end.time</td>
<td>The time, expressed in string form, at which the proxy stopped sending the request to the URL specified in the TargetEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC. This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the time-stamp value of 1377112607413.</td>
<td>Target Request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.sent.end.stamp</td>
<td>The timestamp value specifying when the proxy finished sending the request to the URL specified in the TargetEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Target Request</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>target.sent.start.time</td>
<td>The time, expressed in string form, at which the proxy began sending the request to the URL specified in the TargetEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC. This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.</td>
<td>Target Request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.sent.start.timestamp</td>
<td>The timestamp value specifying when the proxy started sending the request to the URL specified in the TargetEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Target Request</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>target.ssl.enabled</td>
<td>Whether TargetEndpoint is running on TLS/SSL</td>
<td>Target Request</td>
<td>Boolean</td>
<td>Read</td>
</tr>
<tr>
<td>target.url</td>
<td>The URL configured in the TargetEndpoint XML file or the dynamic target URL (if target.url is set during the message flow). The variable does not include any additional path elements or query parameters. Returns null if called out of scope or otherwise unset.</td>
<td>Target Request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>variable.expectedcn</td>
<td>Variable exposed for the common name if it’s running on TLS/SSL</td>
<td>Proxy Request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>virtualhost.aliases</td>
<td>Host aliases of the virtual host that is hit during a particular request</td>
<td>Proxy Request</td>
<td>String_array</td>
<td>Read</td>
</tr>
<tr>
<td>virtualhost.name</td>
<td>Name of the virtual host that serves the originating client request</td>
<td>Proxy Request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>virtualhost.ssl.enabled</td>
<td>Returns true if TLS/SSL is enabled in the virtual host configuration</td>
<td>Proxy request</td>
<td>Boolean</td>
<td>Read</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.2.3 System Variables

Information pertaining to the system is described in system variables.

Every system variable consists of two parts, a prefix `system` and a function. For example: `system.time`, `system` is the prefix and `time` is the function. Supported system variables are listed in the System Variables table.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.timestamp</td>
<td>The timestamp value specifying when the request is received from the client at the Proxy Endpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Proxy Request</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>Variables</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>system.time</td>
<td>The time, expressed in string form, at which the proxy received a request from a client at the ProxyEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC. This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, ‘Wed, 21 Aug 2013 19:16:47 UTC’ corresponds to the timestamp value of 1377112607413.</td>
<td>Proxy Request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.year</td>
<td>The year portion of the system.time variable.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.month</td>
<td>The month portion of the system.time variable.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.day</td>
<td>The day of month portion of the system.time variable.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.dayof­week</td>
<td>The day of the week portion of the system.time variable.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.hour</td>
<td>The hour portion of the system.time variable.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.minute</td>
<td>The minute portion of the system.time variable.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.second</td>
<td>The second portion of the system.time variable.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.millisecond</td>
<td>The millisecond portion of the system.time variable.</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>system.time.zone</td>
<td>Timezone of the system.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>system.interface.{interface_name}</td>
<td>IP Address of the system.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>Variables</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------</td>
<td>------------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>system.pod.name</td>
<td>The name of the pod where the proxy is running.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>system.region.name</td>
<td>The name of the data center region where the proxy is running.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>system.uuid</td>
<td>The UUID of the message processor handling the proxy.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>router.uuid</td>
<td>The UUID of the router handling the proxy.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
</tbody>
</table>

### 1.5.1.3.4.2.4 Configuration Variables

Configuration Variables describes the configuration settings.

Supported configuration variables are listed in the Configuration Variables table.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>organization.name</td>
<td>Name of the organization</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>environment</td>
<td>Container for environment.name.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>environment.name</td>
<td>Name of the environment in which the transaction ran</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>apiproxy.name</td>
<td>Name of the api proxy</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>apiproxy.revision</td>
<td>The revision number of an API proxy</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>is.error</td>
<td>Error flag</td>
<td>Proxy request</td>
<td>Boolean</td>
<td>Read</td>
</tr>
<tr>
<td>proxy.name</td>
<td>The name attribute configured for the ProxyEndpoint</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
</tbody>
</table>
1.5.1.3.4.2.5 Error Variables

Supported error variables are listed in the Error Variables table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>Error of type Message, which is a contextual object in the error flow</td>
<td>Error</td>
<td>Message</td>
<td>Read/Write</td>
</tr>
<tr>
<td>error.content</td>
<td>Content of the error</td>
<td>Error</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>error.message</td>
<td>Message associated with an error, whose value is available only before the error Flow is executed.</td>
<td>Error</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>error.status.code</td>
<td>The HTTP status code associated with the error. For example: &quot;400&quot;.</td>
<td>Error</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>error.reason.phrase</td>
<td>The reason phrase associated with the error. For example: &quot;Bad Request&quot;.</td>
<td>Error</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>error.transport.message</td>
<td>Any error of type TransportMessage</td>
<td>Error</td>
<td>Transport_Message</td>
<td>Read</td>
</tr>
<tr>
<td>error.state</td>
<td>State in the Flow where an error occurred</td>
<td>Error</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>error.header.&lt;name&gt;</td>
<td>Get or set the response header.</td>
<td>Error</td>
<td>String</td>
<td>Read/Write</td>
</tr>
</tbody>
</table>

1.5.1.3.4.2.6 TLS/SSL Connection Information Variables

API Management lets you enable one-way and two-way TLS/SSL support for virtual hosts.

When you access an API proxy through a virtual host that supports TLS/SSL, API Management captures information about the TLS connection. You can then access the TLS connection information in an API proxy through flow variables.

The kind of TLS/SSL information captured depends upon whether the virtual host is enabled for one-way or two-way TLS. For example, for one-way TLS, API Management captures information about TLS cipher or TLS protocol used in the TLS connection. For two-way TLS, API Management not only captures the same information as captured for one-way TLS, but also captures information about the client’s certificate (cert). For example, the subject or issuer DN of the client cert, the serial number of the client cert and the client cert in the PEM format.
The following are the list of flow variables that contain TLS connection information and are available for access in the API proxy:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tls.cipher</td>
<td>The cipher used by the TLS/SSL connection.</td>
</tr>
<tr>
<td>tls.protocol</td>
<td>The protocol used by the TLS/SSL connection.</td>
</tr>
<tr>
<td>tls.server.name</td>
<td>The requested SNI server name.</td>
</tr>
<tr>
<td>tls.session.id</td>
<td>The session identifier. This flow variable is available when you set either &lt;ConnectionProperties&gt; or &lt;ClientProperties&gt; to true.</td>
</tr>
</tbody>
</table>

The following are the list of flow variables that contain TSL connection information pertaining to the client’s cert.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tls.client.s.dn</td>
<td>The subject Distinguished Name (DN) of the client cert. This variable enables you to capture information about the subject (individual) being certified, including common name (client.cn), organization (client.organization), organization unit (client.organization.unit), e-mail address (client.email.address), country/region codes (client.country), locality (client.locality) etc.</td>
</tr>
<tr>
<td>tls.client.i.dn</td>
<td>The issuer Distinguished Name (DN) of the client cert.</td>
</tr>
<tr>
<td>tls.client.raw.cert</td>
<td>The client cert in the PEM format.</td>
</tr>
<tr>
<td>tls.client.cert.serial</td>
<td>The serial number of the client cert.</td>
</tr>
<tr>
<td>tls.client.cert.fingerprint</td>
<td>The SHA1 fingerprint of the client cert.</td>
</tr>
<tr>
<td>tls.session.id</td>
<td>The session identifier. This flow variable is available when you set either &lt;ConnectionProperties&gt; or &lt;ClientProperties&gt; to true.</td>
</tr>
</tbody>
</table>

To configure a virtual host to capture the TLS/SSL information, you need to request the API Management operations team to set the following properties to true in the virtual host configuration file:
Virtual Host Property | Description
--- | ---
ConnectionProperties | Set it to true to capture TLS connection information for both one-way and two-way TLS.
ClientProperties | Set it to true to capture additional information for two-way TLS.

Related Information

Flow Variables [page 311]

1.5.1.3.4.2.7 API Proxy Flows Variables

Supported API proxy flow variables are listed in the API proxy flow Variables table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>current.flow.name</td>
<td>The name of the currently executed flow (such as “PreFlow”, “PostFlow”, or the name of a conditional flow).</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>current.flow.description</td>
<td>The description of the currently executed flow.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>proxy.flow.name</td>
<td>The name of the most recently executed ProxyEndpoint flow (such as “PreFlow”, “PostFlow”, or the name of a conditional flow).</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>proxy.flow.description</td>
<td>The description of the most recently executed ProxyEndpoint flow.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.flow.name</td>
<td>The name of the most recently executed TargetEndpoint flow (such as “PreFlow”, “PostFlow”, or the name of a conditional flow).</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
</tbody>
</table>
### Variable Description Scope Type Permission

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>target.flow.description</td>
<td>The description of the most recently executed TargetEndpoint flow.</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
</tbody>
</table>

#### 1.5.1.3.4.2.8 Message Variables

Message variables refer to different message types like request, response, or error depending on the point within the API proxy flow in which they are called.

Supported message variables are listed in the Message Variables table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>A contextual object, with the same value as request in the request Flow or as response in the response Flow or as error in the Error flow.</td>
<td>Proxy request</td>
<td>Message</td>
<td>Read/Write</td>
</tr>
<tr>
<td>message.content</td>
<td>Content of the request, response, or error message</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>message.formparam.{formparam_name}.values</td>
<td>Value of the specified form parameter in the message</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>message.formparam.{formparam_name}.values.count</td>
<td>Count of the values of the specified form parameters in the message</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>message.formparams.count</td>
<td>Count of all form parameters in the message</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>message.formparams.names</td>
<td>Value of all form parameters in the message</td>
<td>Proxy request</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>message.formstring</td>
<td>Value of form string in the message</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>message.header.{header_name}</td>
<td>Gets or sets the value of the specified HTTP header in the message</td>
<td>Proxy request</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>message.reason.phrase</td>
<td>Scope begins: Target response Type: String Permission: Read ReasonPhrase of the response message from target</td>
<td>Proxy request</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>message.status.code</td>
<td>HTTP status code of the response message from target</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>message.header.{header_name}.values</td>
<td>All values of the specified HTTP header name in the message</td>
<td>Proxy request</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>message.header.{header_name}.values.count</td>
<td>Count of the values of the specified HTTP header name in the message</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>message.headers.count</td>
<td>Count of all HTTP headers in the message</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>message.headers.names</td>
<td>Value of all HTTP headers in the message</td>
<td>Proxy request</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>messagelogging.{policy-name}.failed</td>
<td>Failure flag for the referenced Message logging policy</td>
<td>Proxy request</td>
<td>Boolean</td>
<td>Read</td>
</tr>
<tr>
<td>messagelogging.failed</td>
<td>Failure flag for Message logging policy</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>message.query-param.{query-param_name}.values</td>
<td>Value of the specified query parameter in the message</td>
<td>Proxy request</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>message.transport.message</td>
<td>Message of type TransportMessage which is a contextual object</td>
<td>Proxy request</td>
<td>TransportMessage</td>
<td>Read</td>
</tr>
</tbody>
</table>
1.5.1.3.4.2.9 Response Message Variables

Response message variables are used in policies to access message components like the header, the query parameters, form parameters, the source IP address, the HTTP message body.

API proxy applies the received response to a series of policies, depending on the request condition. API proxy can either modify or transform the request. Based on the content of the response variable, policies can either transform or reject the request. Supported response variables are listed in the Response Message Variables table.

Response Message Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Scope</th>
<th>Type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>client.sent.end.time</td>
<td>The time, expressed in string form, at which the proxy finished sending the response from the ProxyEndpoint to the client. For example: Wed, 21 Aug 2013 19:16:47 UTC. This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.</td>
<td>Proxy response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>client.sent.end.time-stamp</td>
<td>The timestamp value specifying when the proxy finished sending the response to the client from the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Proxy response</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>client.sent.start.time</td>
<td>The time, expressed in string form, at which the proxy began sending the response from the ProxyEndpoint to the client. For example: Wed, 21 Aug 2013 19:16:47 UTC.</td>
<td>Proxy response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>client.sent.start.timestamp</td>
<td>The timestamp value specifying when the proxy began sending the response to the client from the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Proxy response</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>message.reason.phrase</td>
<td>ReasonPhrase of the response message from target</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>message.status.code</td>
<td>HTTP status code of the response message from target</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>response</td>
<td>Complete response message returned by target</td>
<td>Target response</td>
<td>Message</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.content</td>
<td>Payload content of the response message returned by the target</td>
<td>Target response</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.formparam.{formparam_name}</td>
<td>The value of a form parameter in the response</td>
<td>Target response</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.formparam.{formparam_name}.values.count</td>
<td>Count of all the values of the specified form parameter in response</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>response.formparams.count</td>
<td>Count of all form parameters in the response</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>------------</td>
</tr>
<tr>
<td>response.formparams.names</td>
<td>The names of all the form parameters in the response</td>
<td>Target response</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>response.header.{header_name}</td>
<td>Gets or sets the value of a specified HTTP header in the response. If the header has multiple values (such as a CSV list), a GET returns the first value only.</td>
<td>Target response</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.header.{header_name}.values</td>
<td>All the values of a specified HTTP header in response</td>
<td>Target response</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>response.header.{header_name}.values.count</td>
<td>Count of all the values of the specified HTTP header in response</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>response.headers.count</td>
<td>Count of all the headers in the response</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>response.headers.names</td>
<td>The names of all the headers in the response</td>
<td>Target response</td>
<td>Collection</td>
<td>Read</td>
</tr>
<tr>
<td>response.reason.phrase</td>
<td>The response reason phrase for a particular request</td>
<td>Target response</td>
<td>String</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.status.code</td>
<td>The response code returned for a request. You can use this variable to override the response status code, which is stored in message.status.code.</td>
<td>Target response</td>
<td>Integer</td>
<td>Read/Write</td>
</tr>
<tr>
<td>response.transport.message</td>
<td>Response of type TransportMessage which is a contextual object</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.country</td>
<td>Country/region of the TLS/SSL certificate presented by the target server</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.email.address</td>
<td>E-mail address of the TLS/SSL certificate presented by the target server</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.organization</td>
<td>Organization of the TLS/SSL certificate presented by the target server</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>target.organization.unit</td>
<td>Organization unit of the TLS/SSL certificate presented by the target server</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.locality</td>
<td>Locality (city) of the TLS/SSL certificate presented by the target server</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.received.end.time</td>
<td>This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, ‘Wed, 21 Aug 2013 19:16:47 UTC’ corresponds to the timestamp value of 1377112607413.</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.received.end.timestamp</td>
<td>The timestamp value specifying when the TargetEndpoint finished receiving the response from the target. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Target response</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>target.received.start.time</td>
<td>The time, expressed in string form, at which the TargetEndpoint finished receiving the response from the target. For example: Wed, 21 Aug 2013 19:16:47 UTC. This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, ‘Wed, 21 Aug 2013 19:16:47 UTC’ corresponds to the timestamp value of 1377112607413.</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.received.start.timestamp</td>
<td>The timestamp value specifying when the TargetEndpoint started receiving the response from the target. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.</td>
<td>Target response</td>
<td>Long</td>
<td>Read</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Scope</td>
<td>Type</td>
<td>Permission</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>target.state</td>
<td>State of the TLS/SSL certificate presented by the target server</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.host</td>
<td>The domain name of the target service returning the response to the API proxy.</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.ip</td>
<td>The IP address of the target service returning the response to the API proxy.</td>
<td>Target response</td>
<td>String</td>
<td>Read</td>
</tr>
<tr>
<td>target.port</td>
<td>The port number of the target service returning the response to the API proxy.</td>
<td>Target response</td>
<td>Integer</td>
<td>Read</td>
</tr>
<tr>
<td>target.scheme</td>
<td>Returns http or https depending on the request message</td>
<td>Target response</td>
<td>String</td>
<td>Read/Write</td>
</tr>
</tbody>
</table>

1.5.1.3.4.2.10 Path Variables

Supported path variables are listed in the Path Variables table.

**Path Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request variables</td>
<td>request.uri</td>
<td>The HTTP request path, which includes a path and a query string separated by a question mark ( ? )</td>
</tr>
<tr>
<td></td>
<td>request.path</td>
<td>The HTTP request path without the query string</td>
</tr>
<tr>
<td></td>
<td>request.querystring</td>
<td>The portion of the HTTP request path after the question mark ( ? )</td>
</tr>
<tr>
<td>Application variables</td>
<td>application.basepath</td>
<td>The deployment base path (specified during API deployment)</td>
</tr>
<tr>
<td>Proxy variables</td>
<td>proxy.basepath</td>
<td>The base path as configured in the proxy XML file.</td>
</tr>
<tr>
<td></td>
<td>proxy.patsuffix</td>
<td>The portion of the request path after the proxy basepath, which is determined by any conditional flow URIs</td>
</tr>
<tr>
<td>Target variables</td>
<td>request.url</td>
<td>The complete URL of the final request made.</td>
</tr>
<tr>
<td></td>
<td>target.basepath</td>
<td>The path (without host or port) in the URL configured in the target XML file or the dynamic target URL (if target.url is set during the message flow)</td>
</tr>
</tbody>
</table>
### 1.5.1.3.4.2.11 Multi-value HTTP Headers in an API Proxy

Access and extract multiple values in an HTTP header.

An HTTP header is a name value pair that allows a client application or a backend system to pass additional information about requests and responses. Following are a few examples of simple http headers:

  The `Retry-After` request header passes the information about how long the service is unavailable to the requesting client.
- **Server: APIM/3.0**
  The `Server` header passes the information about the software used by the origin server to handle the request.

An HTTP header can have multiple values depending on `<header definition list link>`. A multi-valued HTTP header has comma-separated values. Following are a few examples of headers that contain multiple values:

- **Content-Language: en, de, fr**
- **Content-Type: application/json, application/xml, text/html**
- **Cache-Control: no-transform, no-store, proxy-revalidate**
- **Accept: text/*, text/html, text/html;level=1, */**

API Management allows developers to access HTTP headers in policies or in conditional flows using flow variables. Following are a list of variables in API Management that you can use to access a specific HTTP request or response header:

- `request.header.{header_name}`
- `request.header.{header_name}.values`
- `request.header.{header_name}.values.count`
- `request.headers.count`
- `response.header.{header_name}`
- `response.header.{header_name}.values`
- `response.header.{header_name}.values.count`
- `response.headers.count`

Following is a sample AssignMessage policy that shows how to read the value of a request header and store it in a variable:

```
<AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
  <AssignVariable>
    <Name>var</Name>
  </AssignVariable>
</AssignMessage>
```
Accessing Multiple Values in an HTTP Header

Accessing the values of HTTP headers in API Management policies wherein only the first value of the header is returned is incorrect. The approach of extracting only one value of an HTTP header can lead to issues if the specific header has more than one value. Following are a few examples of multi-value header access:

- **Example 1**: Read a multi-valued header using javascript code
  Consider that the `Accept` header has multiple values as shown below:
  ```
  Accept: application/xml, text/html, application/xhtml+xml
  ```
  Following is sample JavaScript code that reads the value of Accept header:
  ```
  var valuesofacceptheader = context.getVariable("request.header.Accept");
  ```
  The above sample code returns only the first value of the `Accept` header, which is `application/xml`.

- **Example 2**: Read a multi-valued header in Assign Message Policy
  Consider that the `Access-Control-Allow-Headers` header has multiple values as shown below:
  ```
  Access-Control-Allow-Headers: content-type, authorization
  ```
  Following is the sample Assign message policy payload that sets the value of `Access-Control-Allow-Headers` header:
  ```
  <AssignMessage async="false" continueOnError="false" enabled="true" xmlns='http://www.sap.com/apimgmt'>
    <Headers>
      <Header name="Access-Control-Allow-Headers">{request.header.Access-Control-Allow-Headers}</Header>
    </Headers>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
    <AssignTo createNew="true" transport="http" type="response"/>
  </AssignMessage>
  ```
  The above sample code sets the `Access-Control-Allow-Headers` header with only the first value, which is `content-type`.

In both the examples above, only the first value of the multi-valued headers are returned. Later, if any other policy in an API proxy tries to use these values to perform some function, then it could lead to errors.

Extracting Multiple Values in an HTTP Header

To extract multiple values in an HTTP header, you can use the relevant built-in flow variables such as `request.header.{header_name}.values.count`, `request.header.{header_name}.values`, `response.header.{header_name}.values.count`, and `response.header.{header_name}.values`. 
You can then iterate to retrieve all the values from a specific header using a sample JavaScript code as shown below:

```javascript
for (var i = 1; i <= context.getVariable('request.header.Content-Type.values.count'); i++) {
    print(context.getVariable('request.header.Content-Type' + i));
}
```

### 1.5.1.3.5 File Resource

File resource is a script or code snippet that can be attached to flows using policies.

An API proxy container supports definition of a number of Java, Python, or XSL scripts. These scripts can be executed in the context of either a java script, python script, or XSL transformation policy. Once a script is defined, it can be applied as a either a java script, python script, or XSL transformation policy in different flows.

### 1.5.1.3.6 API Proxy Structure

During an import or export of an API proxy, the API proxy follows a specific predefined structure.

The structure of an API proxy is as follows:

**API Proxy Structure**

<table>
<thead>
<tr>
<th>Folder Name</th>
<th>Path</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIProxy</td>
<td>\API Proxy</td>
<td>Root folder that contains the APIProxyEndPoint, APITargetEndPoint, APIResource, Documentation, FileResource, and Policy information.</td>
</tr>
<tr>
<td>APIProxyEndPoint</td>
<td>\API Proxy\APIProxyEndPoint</td>
<td>This folder has a <code>&lt;Proxy Endpoint name&gt;.xml</code> contains one file for every resource associated with the APIProxy. Each file that contains information about Proxy Endpoint, resources, documentation, and the policy assignments on the proxy endpoint stream.</td>
</tr>
</tbody>
</table>
### Folder Name | Path | Contents
---|---|---
APIResource | \API Proxy\APIResource\ | Contains one file for every resource associated with the APIProxy. Each <API Resource name>.xml file contains the API resource details such as resource name, title, documentation, and so on.

Documentation | \API Proxy\Documentation | Contains one documentation file for every resource. Each document follows the naming convention <APIResource name>_<language>.html. For example, if the API Resource name is PurchaseOrder and the supported language is English, then the document file name is PurchaseOrder_en.html. The file provides the documentation content relevant to the associated resource.

FileResource | \API Proxy\FileResource | Lists all the scripts attached to the policy. Only Java, Python, and XSL Scripts are supported. Follow the below naming convention:
- Java Script: <JavaScript name>.js
- Python script: <PythonScript name>.py
- XSL script: <XSLScript name>.xsl

Policy | \API Proxy\Policy | Contains a list of all policies attached to the API Proxy. Each policy is available as a separate file with the naming convention <Policy name>.xml. The folder should also contain a defaultRaiseFaultPolicy.xml.

<APIProxyName>.xml | \API Proxy | <APIProxyName>.xml | This file contains the header information of the proxy endpoint, target endpoint, policies, and file resources.

---

### Note
- When you import an API proxy, ensure that the API proxy name in the <APIProxy name>.xml file and the value of the base_path field (inside the APIProxyEndpoint file) is unique.
- If the API proxy does not contain any API resources, then the APIResources, Documentation, FileResource, and Policy folders are not required in the .zip file during import of API proxy.
Route

A proxy endpoint can connect to one or more Target Endpoints. A route connects the proxy endpoint to the Target Endpoint. It determines which Target Endpoint to invoke based on the proxy endpoint configuration.

**Note**

You can also have an empty route, which means you do not define a Target Endpoint. You can define such routes when you do not want to forward the request message to any Target Endpoint. For example, flows that generate OAuth token.

Route Rule

All requests are forwarded to the respective Target Endpoints by implementing certain rules on the route. The Route Rule is basically a conditional statement that determines the Target Endpoint. When more than one Target Endpoint is available, the Route Rule evaluates the condition and, if true, the request is forwarded to the named target endpoint.

For more information on how to define multiple target endpoints using Route Rule, see Enable Dynamic Routing [page 360]

Fault Rule

A lot of error conditions can arise when API proxies are servicing requests from applications. For example, you may encounter network issues when communicating with backend services, applications may present expired credentials, request messages may be incorrectly formatted, and so on. In such cases, it is important to handle these errors in a customized manner. When an API proxy encounters an error, the default behavior is to exit from the normal processing pipeline and to enter an error Flow. This error Flow bypasses any remaining processing Steps and Policies. As a result the raw error message or codes are returned to the requesting application. It is important to modify this behavior and improve usability. The API Platform enables you to customize exception handling by defining Fault Rules. Fault Rules can be attached to proxy endpoints, target endpoints, and Route rules. A Fault Rule is an XML configuration element that specifies:

- A condition that classifies a fault based on the predefined category, subcategory, or name of the fault
- One or more Policies that define the behavior of the FaultRule

1.5.1.4 Endpoint Property Reference

This topic describes transport properties that can be set in TargetEndpoint and ProxyEndpoint configurations to control messaging and connection behavior.

**Proxy Endpoint Properties [page 355]**

ProxyEndpoint HTTPTargetConnection elements define a set of HTTP transport properties. These properties can be used to set transport-level configurations.

**Target Endpoint Properties [page 356]**

HTTP transport properties configured in the HTTPTargetConnection element in TargetEndpoint configurations defines a set of properties to set transport level configurations.
1.5.1.4.1 Proxy Endpoint Properties

ProxyEndpoint HTTPTargetConnection elements define a set of HTTP transport properties. These properties can be used to set transport-level configurations.

You can export the proxy endpoint as a zip file and add the ProxyEndpoint properties to the the APIProxyEndPoint subfolder under the APIProxy parent folder. For more information, see Export an API [page 450].

Properties are set on ProxyEndpoint HTTPProxyConnection elements as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ProxyEndPoint default="true">
  <name>default</name>
  <base_path>/sdf</base_path>
  <properties>
    <property>
      <name>temp</name>
      <value>v1</value>
    </property>
  </properties>
  <routeRules>
    <routeRule>
      <name>default</name>
      <targetEndPointName>default</targetEndPointName>
      <sequence>1</sequence>
      <faultRules/>
    </routeRule>
  </routeRules>
  <faultRules/>
  <preFlow>
    <name>PreFlow</name>
  </preFlow>
  <postFlow>
    <name>PostFlow</name>
  </postFlow>
  <conditionalFlows/>
</ProxyEndPoint>
```

*iNote*

Alternatively, you can navigate to the Develop tab, choose the API proxy that you deployed from the APIs list, and select the Proxy Endpoint tab. In the Proxy Endpoint Properties section, choose Add and enter the Property Name and the Value as defined in the ProxyEndpoint Transport Property Specification table.

### ProxyEndpoint Transport Property Specification

<table>
<thead>
<tr>
<th>Property Value</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Forwarded-For</td>
<td>false</td>
<td>On setting it to true, the virtual host’s IP address is added to the outbound request as the value of the HTTP X-Forwarded-For header.</td>
</tr>
<tr>
<td>Property Value</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>request.streaming.enabled</td>
<td>false</td>
<td>Default value (false): HTTP request payloads are read into a buffer, and policies operating on the payload work as expected. True: HTTP request payloads are not read into a buffer, they are streamed to the TargetEndpoint request flow. And, policies operating on the payload in the ProxyEndpoint request flow are bypassed. For more information about enabling streaming, see Enable Streaming of Requests and Responses in an API Proxy [page 366].</td>
</tr>
<tr>
<td>response.streaming.enabled</td>
<td>false</td>
<td>Default value (false): HTTP response payloads are read into a buffer, and policies operating on the payload work as expected. True: HTTP response payloads are not read into a buffer, they are streamed to the TargetEndpoint request flow. And, policies operating on the payload in the ProxyEndpoint response flow are bypassed. For more information about enabling streaming, see Enable Streaming of Requests and Responses in an API Proxy [page 366].</td>
</tr>
</tbody>
</table>
| compression.algorithm   | N/A           | Supported values:  
  - gzip: always send message using gzip compression  
  - deflate: always send message using deflate compression  
  - none: always send message without any compression  
For example, when a client submits a request that uses gzip compression, API Management forwards the request to target using gzip compression. You can configure compression algorithms to be explicitly applied by setting this property on the TargetEndpoint or ProxyEndpoint. |
| api.timeout             | N/A           | Configure the timeout for individual API proxies.                                                                                                                                                                                                                                                                                           |

Parent topic: Endpoint Property Reference [page 354]

Related Information

Target Endpoint Properties [page 356]

1.5.1.4.2 Target Endpoint Properties

HTTP transport properties configured in the HTTPTargetConnection element in TargetEndpoint configurations defines a set of properties to set transport level configurations.

You can export the proxy endpoint as a zip file and add the TargetEndpoint properties to the the APITargetEndPoint subfolder under the APIProxy parent folder. For more information, see Export an API [page 450].
Configure the properties on TargetEndpoint HTTP TargetConnection elements:

**Sample Code**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TargetEndPoint>
  <name>default</name>
  <url>http://www.abc.com</url>
  <provider_id>NONE</provider_id>
  <isDefault>true</isDefault>
  <properties>
    <property>
      <name>temp</name>
      <value>value</value>
    </property>
    <property>
      <name>test</name>
      <value>v2</value>
    </property>
  </properties>
  <faultRules/>
  <preFlow>
    <name>PreFlow</name>
  </preFlow>
  <postFlow>
    <name>PostFlow</name>
  </postFlow>
  <conditionalFlows/>
</TargetEndPoint>
```

**i Note**

Alternatively, you can navigate to the Develop tab, choose the API proxy that you deployed from the APIs list, and select the Target Endpoint tab. In the Target Endpoint Properties section, choose Add and enter the Property Name and the Value as defined in the TargetEndpoint Transport Property Specification table.

### TargetEndpoint Transport Property Specification

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keepalive.timeout.millis</td>
<td>60000 milliseconds</td>
<td>Connection idle timeout for the target connection in the connection pool. If the connection in the pool is idle beyond the specified limit, then the connection is closed.</td>
</tr>
<tr>
<td>connect.timeout.millis</td>
<td>3000 milliseconds</td>
<td>Target connection timeout. returns an HTTP 503 status code if a connection timeout occurs.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| io.timeout.millis | 55000 milliseconds | If there is no data to read for the specified number of milliseconds, or if the socket is not ready to write data for specified number of milliseconds, then the transaction is treated as a timeout.  
* If a timeout happens while writing the HTTP request, 408, Request Timeout is returned.  
* If a timeout happens while reading the HTTP response, 504, Gateway Timeout is returned.  

**Note**  
Timeout value cannot exceed the default value of 55000 milliseconds. However, if you want to change the default value, then you must raise a request to API Management Operations team with the required value you want to set for `io.timeout.millis` property. Based on the value that you want to set, the API management Operations team will update the Virtual host’s `proxy_read_timeout` property value. Once the `proxy_read_timeout` value is set by the Operations team, you can change the value of `io.timeout.millis`. The `io.timeout.millis` value must be less than the value of `proxy_read_timeout` value. |
| supports.http10 | true          | If true and the client sends a 1.0 request, the target is also sent a 1.0 request. Otherwise 1.1 request is sent to target.                                                                                                                                                            |
| supports.http11 | true          | If true and the client sends a 1.1 request, the target is also sent a 1.1 request, otherwise 1.0 request is sent to target.                                                                                                                                                           |
| success.codes   | N/A           | By default, HTTP code 4XX or 5XX are treated as errors, HTTP code 1XX, 2XX, 3XX as success. This property enables explicit definition of success codes, for example, 2XX, 1XX, 505 treats any 100, 200 and 505 HTTP response codes as success.  
Setting this property overwrites the default values. Therefore, if you want to add HTTP code 400 to the list of default success codes, set this property as:  
```xml  
<Property name="success.codes">1xx,2xx,3xx,400</Property>  
```

If you want only HTTP code 400 to be treated as a success code, set the property as:  
```xml  
<Property name="success.codes">400</Property>  
```

By setting HTTP code 400 as the only success code, the codes 1xx, 2xx, and 3xx are treated as failures.  
| compression.algorithm | N/A            | Supported values:  
* gzip: always send message using gzip compression  
* deflate: always send message using deflate compression  
* none: always send message without any compression  
For example, when a client submits a request that uses gzip compression, API Management forwards the request to target using gzip compression. You can configure compression algorithms to be explicitly applied by setting this property on the TargetEndpoint or ProxyEndpoint. |
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable.method.override</td>
<td>false</td>
<td>For the specified HTTP method, sets an X-HTTP-Method-Override header on the outbound request to the target service. For example, <code>&lt;Property&gt;&lt;name&gt;=&quot;GET.override.method&quot;&lt;/name&gt;&lt;value&gt;POST&lt;/value&gt;&lt;/Property&gt;</code>.</td>
</tr>
<tr>
<td>request.streaming.enabled</td>
<td>false</td>
<td>Default value (false): HTTP request payloads are read into a buffer, and policies operating on the payload work as expected. True: HTTP request payloads are not read into a buffer, they are streamed to the target endpoint. And, policies that operate on the payload in the TargetEndpoint request flow are bypassed. For more information about enabling streaming, see [Enable Streaming of Requests and Responses in an API Proxy](page 366)</td>
</tr>
<tr>
<td>*.override.method</td>
<td>N/A</td>
<td>For the specified HTTP method, sets an X-HTTP-Method-Override header on the outbound request. For example, <code>&lt;Property&gt;&lt;name&gt;=&quot;GET.override.method&quot;&lt;/name&gt;&lt;value&gt;POST&lt;/value&gt;&lt;/Property&gt;</code>.</td>
</tr>
<tr>
<td>response.streaming.enabled</td>
<td>false</td>
<td>Default value (false): HTTP response payloads are read into a buffer, and policies that can operate on the payload work as expected. True: HTTP response payloads are not read into a buffer; they are streamed as-is to the ProxyEndpoint response flow. In this case, any policies that operate on the payload in the TargetEndpoint response flow are bypassed. For more information about enabling streaming, see [Enable Streaming of Requests and Responses in an API Proxy](page 366)</td>
</tr>
<tr>
<td>request.retain.headers.enabled</td>
<td>true</td>
<td>By default, all HTTP headers on outbound messages are retained. On setting it to true, all HTTP headers present on the inbound request are set on the outbound request.</td>
</tr>
<tr>
<td>request.retain.headers</td>
<td>N/A</td>
<td>Set HTTP headers from the request on the outbound request to the target service. For example, to ‘passthrough’ the User-Agent header, set the value of request.retain.headers to User-Agent. Specify multiple HTTP headers as a comma-separated list, for example, User-Agent,Referer,Accept-Language. This property overrides request.retain.headers.enabled. If request.retain.headers.enabled is set to false, any headers specified in the request.retain.headers property are still set on the outbound message.</td>
</tr>
<tr>
<td>response.retain.headers.enabled</td>
<td>true</td>
<td>By default, all HTTP headers on outbound messages are retained. On setting it to true, all HTTP headers present on the inbound response from the target service are set on the outbound response before it is passed to the ProxyEndpoint.</td>
</tr>
<tr>
<td>response.retain.headers</td>
<td>N/A</td>
<td>Set HTTP headers from the response on the outbound response before it is passed to the ProxyEndpoint. For example, to passthrough the Expires header, set the value of response.retain.headers to Expires. Specify multiple HTTP headers as a comma-separated list, for example, Expires,Set-Cookie. This property overrides response.retain.headers.enabled. If response.retain.headers.enabled is set to false, any headers specified in the response.retain.headers property are still set on the outbound message.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>retain.queryparams</td>
<td>N/A</td>
<td>Set query parameters on the outbound request. For example, to include the query parameter apikey from the request message, set retain.queryparams to apikey. Specify multiple query parameters as a comma separated list, for example, apikey, environment. This property overrides retain.queryparams.enabled.</td>
</tr>
<tr>
<td>retain.queryparams.enabled</td>
<td>true</td>
<td>By default, all query parameters on outbound requests are retained. On setting it to true, all query parameters present on the inbound request are set on the outbound request to the target service.</td>
</tr>
</tbody>
</table>

**Parent topic:** Endpoint Property Reference [page 354]

**Related Information**

Proxy Endpoint Properties [page 355]

### 1.5.1.5 Enable Dynamic Routing

Define route rules to enable dynamic routing in an API proxy.

**Context**

A route connects an API proxy endpoint to an API target endpoint. It governs the path of a request from proxy endpoint to target endpoint and determines which target endpoint to invoke based on the condition defined in proxy endPoint definition. Typically, a route includes a URL used to access the API proxy endpoint and a URL of the backend service defined in target endpoint definition.

In API Management, when you create an API proxy, a default route rule is set and it always forwards the request to the default target endpoint defined in target endpoint definition. When more than one target endpoint is defined, the route rule evaluates the condition set in proxy endpoint definition. If the condition evaluates to true, it forwards the request to the named target endpoint.

The following procedure describes how to achieve dynamic routing in API Management. Let’s say you want to route an API proxy request to two different target endpoints, a default target endpoint and a new target endpoint based on a condition set in the proxy endpoint definition.

For our implementation, let’s consider the following two target endpoints:

- **Target_Endpoint_1 (default)**
  
  https://services.odata.org/V2/Northwind/Northwind.svc/

- **Target_Endpoint_2**
  
  https://services.odata.org/V2/OData/OData.svc/
Procedure

Creating a simple API proxy
1. Navigate to API portal.
2. Choose the navigation icon on the left and choose Develop.
3. Under APIs, choose Create to create a simple API proxy.
4. In the Create API wizard, choose the URL radio button.

   i Note
   You can also choose to create an API by choosing the API Provider option. For more information, see Create an API [page 393]

5. In the URL field, enter the target URL of your backend service. In this case, URL pointing to Target_Endpoint_1 (default).
6. Enter a name and a title for your API proxy. In this case, let’s enter the API proxy name as Dynamic_Routing.
7. Scroll down the wizard and enter the base path of your API proxy in the API Base Path field. In this case, let’s enter the base path as /multitargets.
8. Choose Create.
9. Save and deploy your API proxy.

   i Note
   When the API proxy is created, the default route rule is set. It points to the default target endpoint and no rule is attached to it.

Steps for defining new target endpoint
10. Navigate to the Develop tab. From the APIs list, choose the API proxy that you deployed.
11. Download the newly deployed API proxy using the Export option. For more information, see Export an API [page 450]

   A zip file called Dynamic_Routing.zip is downloaded.

   A parent folder called APIProxy is created. The APIProxy folder consists of various subfolders and files. For more information, see API Proxy Structure [page 352].
13. Open the APITargetEndPoint subfolder.

   You see a file named default.xml. The default.xml file contains the URL of the default target endpoint.
14. Create a new XML file named Target_EndPoint_2.xml with the following content. In the Target_EndPoint_2.xml file, you need to enter a name and the URL of the new target endpoint to which the request must be routed dynamically.

   i Note
   The <isDefault> attribute must be set to false for all the new target endpoints that you define. Whereas, for the default target endpoint the <isDefault> attribute would by default be set to true.
You see two files named `default.xml` and `Target_EndPoint_2.xml` in the `APITargetEndPoint` subfolder.

### Steps for defining conditions using Route Rule

15. Open the `APIProxyEndPoint` subfolder.

You see a file named `default.xml` file.


The `default.xml` file contains information about your API proxy such as base path, flows, policies and, the default route rule.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ProxyEndPoint default="true">
  <name>default</name>
  <base_path>/multitargets</base_path>
  <properties/>
  <routeRules>
    <routeRule>
      <name>default</name>
      <targetEndPointName>default</targetEndPointName>
      <sequence>1</sequence>
      <faultRules/>
    </routeRule>
  </routeRules>
  <faultRules/>
  <preFlow>
    <name>PreFlow</name>
  </preFlow>
  <postFlow>
    <name>PostFlow</name>
  </postFlow>
  <conditionalFlows/>
</ProxyEndPoint>
```

17. Update the value of the `<sequence>` attribute to 2.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<routeRule>
```

---

SAP API Management

SAP API Management in the Cloud Foundry Environment
The resulting default.xml file must reflect the following content.

### Sample Code

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ProxyEndPoint default="true">
  <name>default</name>
  <base_path>/multitargets</base_path>
  <properties/>
  <routeRules>
    <routeRule>
      <name>default</name>
      <targetEndPointName>default</targetEndPointName>
      <sequence>2</sequence>
      <faultRules/>
    </routeRule>
    <routeRule>
      <name>Target_EndPoint_2</name>
      <targetEndPointName>Target_EndPoint_2</targetEndPointName>
      <sequence>1</sequence>
      <faultRules/>
    </routeRule>
    <faultRules/>
    <preFlow>
      <name>PreFlow</name>
    </preFlow>
    <postFlow>
      <name>PostFlow</name>
    </postFlow>
    <conditionalFlows/>
  </routeRules>
</ProxyEndPoint>
```

18. Define a new route rule named Target_EndPoint_2 by adding the following content to the default.xml file.

### Sample Code

```xml
<routeRule>
  <name>Target_EndPoint_2</name>
  <targetEndPointName>Target_EndPoint_2</targetEndPointName>
  <sequence>1</sequence>
  <faultRules/>
</routeRule>
```

The resulting default.xml file must reflect the following content.

### Sample Code

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ProxyEndPoint default="true">
  <name>default</name>
  <base_path>/multitargets</base_path>
  <properties/>
  <routeRules>
    <routeRule>
      <name>Target_EndPoint_2</name>
      <targetEndPointName>Target_EndPoint_2</targetEndPointName>
      <sequence>1</sequence>
      <faultRules/>
    </routeRule>
  </routeRules>
</ProxyEndPoint>
```
19. Define a condition based on which you want to route the request dynamically. In this case, let's add a `proxy.pathsuffix MatchesPath` condition under the `Target_EndPoint_2` Route Rule and set it to the path called `/Categories`.

**Sample Code**

```xml
<routeRule>
  <name>Target_EndPoint_2</name>
  <conditions>proxy.pathsuffix MatchesPath "/Categories"</conditions>
  <targetEndPointName>Target_EndPoint_2</targetEndPointName>
  <sequence>1</sequence>
  <faultRules/>
</routeRule>
```

The resulting `default.xml` file must reflect the following content.

**Sample Code**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ProxyEndPoint default="true"/>
  <name>default</name>
  <base_path>/multitargets</base_path>
  <properties/>
  <routeRules>
    <routeRule>
      <name>Target_EndPoint_2</name>
      <conditions>proxy.pathsuffix MatchesPath "/Categories"</conditions>
      <targetEndPointName>Target_EndPoint_2</targetEndPointName>
      <sequence>1</sequence>
      <faultRules/>
    </routeRule>
    <routeRule>
      <name>default</name>
      <targetEndPointName>default</targetEndPointName>
      <sequence>2</sequence>
      <faultRules/>
    </routeRule>
  </routeRules>
  <faultRules/>
  <preFlow>
    <name>PreFlow</name>
  </preFlow>
  <postFlow>
    <name>PostFlow</name>
  </postFlow>
  <conditionalFlows/>
</ProxyEndPoint>
```
If you have defined more than one route rule in the proxy endpoint as shown in the above codeblock, their sequence in the XML configuration is important. The first Route Rule to match gets executed. (Route rules with no condition always match). In the above codeblock, if the default route rule appeared first, it would be executed even if the condition of the Target_EndPoint_2 route rule would have matched. Hence, it is always recommended to list your conditional route rules before an unconditional route rule.

Steps for viewing dynamic routing

21. Add the new target endpoint name that you defined.

Sample Code

```xml
<targetEndPoints>
   <targetEndPoint>Target_EndPoint_2</targetEndPoint>
   <targetEndPoint>default</targetEndPoint>
</targetEndPoints>
```

22. Compress the APIProxy parent folder.
23. Navigate to API portal and import the compressed APIProxy.zip file. For more information, see Import an API [page 448].
24. Choose the imported API proxy.
25. Under Proxy EndPoint tab, in the Route Rules section, you must see two Route Rules that you defined earlier.

You can also add route conditions directly in API portal User Interface instead of adding it manually in the API proxy EndPoint definition file as shown in step 19.

26. Click on the API proxy URL.

The request must be routed to the default target endpoint.
27. Append /Categories to the API proxy URL in your browser.

The request must be routed dynamically to the new target endpoint.
To validate the response, copy and paste the actual URL of the backend service with path suffix /Categories.

The response obtained must match the response obtained in step 27.

All the policies that you attach in the target endpoint via the API portal user interface are applied only to the default target endpoint. In case, if you need to enforce policies on the non-default target endpoint, then you must import the API proxy bundle and manually add the policies in the required target endpoint definition file.
1.5.1.6 Enable Streaming of Requests and Responses in an API Proxy

Configure an API proxy to enable HTTP request and response streaming.

Context

By default, API Management disables streaming of request and response payloads. The payloads are stored directly into a buffer before API proxy pipeline picks them for processing. With streaming disabled, the policies that operate on the payloads work as expected. You can alter this behavior by enabling streaming. When you have enabled streaming, the API proxy pipeline processes the request and response payloads as-is and streams them without any modifications to the client application and to the target endpoint.

You can enable streaming if your API proxy handles large request and response payloads. In API Management, message payload size is restricted to 10 MB for non-streamed HTTP requests and responses. For streamed requests and responses, the message payload size is restricted to 500 MB. For more information on request and response payload size, see .

Note

When you have enabled streaming, API Management recommends that you don’t attach policies that require access to the request and response payloads. These policies can cause errors or initiate buffering, which limits the payload size and hence defeats the purpose of enabling streaming for handling large payloads. You can attach policies such as Authentication or message logging policies since these policies don’t interact with the request and response payloads.

Procedure

1. To enable request streaming, in your API proxy bundle, add the request.streaming.enabled property in the proxy endpoint and target endpoint definitions and set it to true.
2. To enable response streaming, in your API proxy bundle, add the response.streaming.enabled property in the proxy endpoint and target endpoint definitions and set it to true.

You can locate the proxy endpoint and target endpoint definition files in your API proxy bundle under APIProxy/APIProxyEndpoint and APIProxy/APITargetEndPoint, respectively.

The following sample code shows how to enable both request and response streaming in the target endpoint definition:
The following sample code shows how to enable both request and response streaming in the proxy endpoint definition:

```
<TargetEndPoint>
  <name>default</name>
  <url>https://www.concursolutions.com/api/v3.0/</url>
  <provider_id>NONE</provider_id>
  <isDefault>true</isDefault>
  <properties>
    <property>
      <name>request.streaming.enabled</name>
      <value>true</value>
    </property>
    <property>
      <name>response.streaming.enabled</name>
      <value>true</value>
    </property>
  </properties>
</TargetEndPoint>
```

You can also enable streaming directly from the API portal as follows:

**Enabling Streaming via API portal**

1. In the API portal, navigate to the **Develop** tab.
2. Select the API for which you want to enable streaming.
3. Choose **Edit** from the top-right corner of the screen.
4. Choose **Proxy Endpoint**.
5. Under **Proxy Endpoint Properties**, choose **Add**.
6. Enter the name of the properties you want to add. In this case, add the `request.streaming.enabled` and `response.streaming.enabled` properties and set their value to `true`.
7. Choose **Target Endpoint**.
8. Under **Target Endpoint Properties**, choose **Add**.
9. Enter the name of the properties you want to add. In this case, add the `request.streaming.enabled` and `response.streaming.enabled` properties and set their value to `true`.
10. Save the changes.
1.5.1.7 Handling URL Redirects in an API Proxy Using Policies

**Symptom**: The API proxy URL redirects to a different target endpoint.

This topic explains how to handle URL redirects in an API proxy using policies.

Let’s say you’ve deployed a simple API proxy, wherein the target endpoint is pointing to the backend service located at the following URL:

https://services.odata.org/V2/Northwind/Northwind.svc

For more information on creating an API proxy, see Create an API Proxy [page 390].

When you click on the API proxy URL, if your browser displays the service URL instead of the proxied URL, it indicates that the service URL that you are trying to access has been moved temporarily or permanently to a new location. When this scenario occurs, API Management doesn’t display the proxied URL. It instead displays the redirected URL of the backend service you provided.

You can handle this URL redirects by adding policies to your API proxy. These policies determine how URL direction is handled within API Management.

In this illustration, we provide two approaches for handling URL redirects. One, wherein we use a Fault Raise policy to stop redirection when the backend service you’re trying to access is moved temporarily to a new location, and additionally send a response to the client indicating what is the new redirected URL of the service. Two, wherein we use a combination of various policies such as, Extract Variables policy, Service Callout policy, and Assign Message policy to gracefully handle the URL redirection without user intervention.

**Stop URL Redirection Using Fault Raise Policy**

Add a Fault Raise policy to your API proxy with the following configuration. Add this policy in the target endpoint (PostFlow/Outgoing Response).

```
<RaiseFault async="true" continueOnError="false" enabled="true" xmlns="http://www.sap.com/apimgmt">
  <!-- Defines the response message returned to the requesting client -->
  <FaultResponse>
    <Set>
      <!-- Sets or overwrites HTTP headers in the response message -->
      <Headers/>
      <!-- Payload contentType="text/plain"--> Sorry for the inconvenience!
      <!-- The target URL that you have provided has been redirected to -->
      <!-- The URL redirection has been stopped due to -->
      <!-- security reasons. Please provide a target URL that doesn't redirect. -->
      <Payload> <500</StatusCode> <!-- sets the reason phrase of the response -->
        <ReasonPhrase>Server Error</ReasonPhrase>
      </Payload>
    </Set>
    <!-- IgnoreUnresolvedVariables=true -->
  </FaultResponse>
</RaiseFault>
```
Add a condition to indicate when this policy must execute. In our illustration, we want to execute this policy only when the backend service URL that you provided is being redirected to a new URL. We achieve this by including the following condition string:

```
response.status.code=307
```

The above condition string indicates that when the http response code received from a backend service is 307, the policy gets executed. The http response code 307 indicates that the service/resource you want access can’t be accessed from its canonical location, but can be accessed from a new temporary location.

If the above policy executes, the client would get the following response, where `{response.header.location}` would actually be replaced with the redirected URL of the backend service you provided.

```
Sorry for the inconvenience!
The target URL that you have provided has been redirected to 
"{response.header.Location}". The URL redirection has been stopped due to 
security reasons. Please provide a target URL that doesn't redirect.
```

Similarly, if the backend service that you want to access is permanently moved to a new location, then you change the condition string statement to `response.status.code=302`.

**Handle URL Redirection Gracefully**

Add an Extract Variables policy to your API proxy with the following configuration. Add this policy to the proxy endpoint (PreFlow/Outgoing Response).

```
<ExtractVariables async="true" continueOnError="false" enabled="true"
xmlns='http://www.sap.com/apimgmt'>
  <Source>response</Source>
  <Header name="Location">
    <Pattern ignoreCase="true">https://{redirectUrl}/{redirectUrlPath}</Pattern>
  </Header>
</ExtractVariables>
```

Add a condition to indicate when this policy must execute. In our illustration, we want to execute this policy only when the backend service URL that you provided is being redirected to a new URL. We achieve this by including the following condition string:

```
response.status.code=307
```
If the above policy executes, the value of the Location header tagged to the response is extracted and stored in the `redirectUrl` and `redirectUrlPath` variables. For example, if the backend service URL (redirected URL) you’re trying to access is `https://services.odata.org/V2/Northwind/Northwind.svc/`, then the value `services.odata.org` is stored in the `redirectUrl` variable and the value `V2/Northwind/Northwind.svc/` is stored in the `redirectUrlPath` variable.

Use the extracted `redirectUrl` and `redirectUrlPath` variables in the Service Callout policy as described further.

Add a Service Callout policy to your API proxy with the following configuration. Add this policy to the proxy endpoint (PreFlow/Outgoing Response).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ServiceCallout async="false" continueOnError="false" enabled="true"
 xmlns="http://www.sap.com/apimgmt">
  <Request clearPayload="true" variable="myRequest">
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  </Request>
  <Response>calloutResponse</Response>
  <HTTPTargetConnection>
    <URL>https://{redirectUrl}/{redirectUrlPath}</URL>
  </HTTPTargetConnection>
</ServiceCallout>
```

Add a condition to indicate when this policy must execute. In our illustration, we want to execute this policy only when the backend service URL that you provided is being redirected to a new URL. We achieve this by including the following condition string:

```
response.status.code=307
```

If the above policy executes, the response message received from the external service indicated in the `URL` element of the policy, is stored in the `calloutResponse` variable. For example, if the values stored in `redirectUrl` and `redirectUrlPath` variables are `services.odata.org` and `V2/Northwind/Northwind.svc/`, respectively, then the response from the URL (redirected URL) `https://services.odata.org/V2/Northwind/Northwind.svc/` will be stored in the `calloutResponse` variable.

In the next step, send the response message obtained from the service callout policy to the client using the Assign Message policy as shown further.

Add an Assign Message policy to your API proxy with the following configuration. Add this policy to the proxy endpoint (PreFlow/Outgoing Response).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AssignMessage async="false" continueOnError="false" enabled="true"
 xmlns='http://www.sap.com/apimgmt'>
  <Set>
    <Payload>{calloutResponse.content}</Payload>
  </Set>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" type="response">response</AssignTo>
</AssignMessage>
```
If the above policy executes, the client receives the response message stored in the `calloutResponse` variable.

Save the API proxy after adding the above policies. You can also create a policy template with the above policies so that you can easily attach the policy template to your API proxies to gracefully handle URL redirection. For more information on policy templates, see `Create a Policy Template [page 453]`.

### 1.5.1.8 Manage Certificates

By using certificates, you can ensure that whenever a call is made to your API, there is a certificate attached to it which confirms the identity of the caller and only if you recognize this identity the API can be processed and return data can be provided.

#### Prerequisites

In this procedure, we assume that you are aware of the process of creating certificates using any tool of your own choice.

#### Context

You need to create key store and trust store certificates to configure 2-way SSL (Secure Sockets Layer). SSL is the standard security technology for establishing an encrypted link between a web server and a web client, such as a browser or an app. An encrypted link ensures that all data passing between the server and the client remains private. To use SSL, a client makes a secure request to the server by using the encrypted `https://` protocol, instead of the unencrypted `http://` protocol. In API Management, you can associate the certificates with the API Provider at the time of API provider registration. This process provides more secure way to access API provider.

Whenever the existing certificate expires, you can upload a new certificate and associate the same with the API provider. You cannot upload an expired certificate.

#### Procedure

1. Log on to API portal.
2. Choose the navigation icon on the top-left and choose `Configure`.
3. Select the `Certificates` tab.
4. Choose `Create`.
5. Select the type of certificate that you want to create.
   - `Trust Store` - A truststore contains certificates used to verify certificates received as part of SSL handshaking. If the certificate received by an SSL client is signed by a valid certificate authority (CA),
then the client makes a request to the CA to authenticate the certificate else self signed certificate can be uploaded in the truststore.

- **Key Store** - A keystore contains an SSL certificate and private key used to validate the server during SSL handshaking.

The examples in this document show the SSL cert and key defined as PEM files, which comply with the X.509 format. If your cert or private key is not defined by a PEM file, you can convert it to a PEM file by using utilities such as openssl. However, many .crt files and .key files are already in the PEM format. If these files are text files, and are enclosed in:

```
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

6. You can either choose to use an existing store or create a new store and then add a new certificate in that store.

7. If you choose to create a new store, then enter the following details: store name, certificate name and appropriate description.

8. If you have chosen to create a key store, then execute the sub-steps below:

   a. Create a JAR file containing your private key, certificate, and a manifest. For example, the JAR file must contain the following files and directories: 

   ```
   /META-INF/descriptor.properties, <main>.pem, <privateKey>.pem
   ```

   ![Name]
   - META-INF
   - main.pem
   - privateKey.pem

   ![Note]
   Ensure to create certificate with unique name. In case if a certificate with the same name exists in your system, then the newly created certificate with overwrite the existing one, and you will loose your old certificate data.

   A keystore JAR can contain only one certificate. If you have a certificate chain, all certs in the chain must be appended into a single PEM file, where the last certificate is signed by a CA. The certs must be appended to the PEM file in the correct order, meaning: 
   ```
   cert  -> intermediate cert(1)  -> intermediate cert(2)  ->  ...  -> root
   ```

   b. In the directory containing your key pair and certificate, create a directory called /META-INF. Then, create a file called descriptor.properties in /META-INF with the following contents:

   ```
   certFile=<main>.pem
   keyFile=<privateKey>.pem
   ```
c. Generate the JAR file containing your key pair and certificate:

```
$ jar -cf myKeystore.jar main.pem privateKey.pem
```

d. Add descriptor.properties to your JAR file:

```
$ jar -uf myKeystore.jar META-INF/descriptor.properties
```

e. Upload the JAR file.

9. If you have chosen to create a trust store, then upload only the PEM file.

10. Choose Create.

Once you create a certificate, you can then associate it with the API provider at the time of API provider registration.

1.5.1.9 API Providers

An API provider defines the connection details for services running on specific hosts whose details you want to access.

Use an API provider to define not only the details of the host you want an application to reach, but also to define any further details that are necessary to establish the connection, for example, proxy settings. You can configure connections to OData-hosted systems from API portal.

**API Providers Connecting to Backend System**

If you want to configure the API Management solution to access data from a server that offers a specific service, for example, an SAP Gateway service, SAP HANA, SAP Process Integration/Process Orchestration, SAP S/4 HANA, or any 3rd party cloud solutions, you can manifest and expose the connection parameters as an API Provider.

**Advantages of Creating API Providers in SAP API Management**

- You can connect to different backend on premise/cloud system.
- Discover services/interfaces.
- Simplifies on-premise connectivity.
- Simplifies configuration if the backend system changes.
Related Information

Create an API Provider [page 374]

1.5.1.9.1 Create an API Provider

Create an API provider to connect to an API provider and browse for APIs.

Prerequisites

- You are assigned the APIPortal/Administrator role.
- The following prerequisite is applicable for Open Connectors type provider only. You have enabled open connector service in the cockpit and noted down the organization secret and user secret.

i Note

To enable open connector service, perform the following:
○ Navigate to **SAP BTP Cockpit > Subscriptions > Open Connectors**

○ Choose **Subscribe**

○ After the service is subscribed, choose **Go to Application** link.

○ You are navigated to Open Connectors home page. In the home page, choose the user icon at the bottom left of the screen and note down the Organization Secret and User Secret.

○ User must have created at least one instance against ORG/USER secret to create an APIProxy in apiportal.

- The following prerequisites are applicable for **On Premise** type Provider only.
  1. Expose the on-premise system using **Cloud Connector**. For more details, see [here](#).
  2. From your **Subaccount**, choose **Cloud Connectors** from the left-hand pane and validate if the system exposed in the previous step is visible in the list.

**Context**

If you want to configure the API Management solution to access data from a server that offers a specific service, for example, SAP Gateway service, SAP HANA, SAP Process Integration/Process Orchestration, SAP S/4 HANA, or any 3rd party cloud solutions, we recommend configuring the solution as an API provider in the connection parameters. The API Management enables you to configure connections to OData-hosted systems.

**Procedure**

1. Log on to the API portal.
2. In the navigation bar on the top-left, choose Choose the navigation icon on the left and choose **Configure**.
   
   The list of API providers appear.
3. Choose **Create**.
4. Enter a name and a description for the API provider.

**i Note**

You cannot use the names mentioned in following list as API provider names:

- DEST_CI
- APIMGMT_PORTAL
- ContentCatalog
- ON_PREM
- Apimgmt_rt
- Apimgmt_rt*
- TC
- TC_*
5. Go to the CONNECT section, enter the required details based on the type of connection you select. Choose:
   - **Internet** to connect to a system on Cloud.
   - **On Premise** to connect to the on-premise system through Cloud Connector.
   - **Open Connectors** to connect third-party APIs via harmonized RESTful APIs.
   - **Cloud Integration** to access all the service endpoints.

### Internet connection type:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Enter a value for the host in the format &quot;example.com&quot;.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter 443 as the port number for SSL and 8080 for all other ports.</td>
</tr>
<tr>
<td>Use SSL</td>
<td>Select the checkbox to secure the connection using SSL (HTTPS protocol).</td>
</tr>
</tbody>
</table>

**i Note**

Setting this option is not sufficient to set up the SSL connection. If you want to use SSL to secure connections to the configured destination, then upload the server certificate in the SAP BTP Cockpit.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trust store certificate</td>
<td>By default, API Management trusts all TLS certificates. To validate a certificate, create a truststore to configure API Management to validate the certificate. Specify the trust store certificate details in the API Provider.</td>
</tr>
<tr>
<td>Key store certificate</td>
<td>Specify the key store certificate details.</td>
</tr>
</tbody>
</table>

### On Premise connection type:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Enter a value for the host in the format &quot;example.com&quot;. Ensure that you provide the cloud connector Virtual Host.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter the port number exposed in the Cloud Connector as per the host.</td>
</tr>
<tr>
<td>Location ID</td>
<td>If you have configured multiple cloud connectors to your SAP BTP account, then provide the location ID of the cloud connector that you want to add.</td>
</tr>
</tbody>
</table>

**i Note**

The Location ID value for a cloud connector is set while configuring the cloud connector.
Choose an authentication method:
- **None**: No Authentication. This is a pass through flow and validation is initiated at the API portal layer.
- **Principal Propagation**: Enables authentication of a message in the receiver system with the same user that issued the message in the corresponding sender system. For more details, see Principal Propagation [page 382].

Select `sap-client` from the dropdown and enter the three-digit client ID. Client-specific data is identified with the client identifier. SAP-client configured in API provider has the highest precedence for runtime calls, metadata fetch, and discovery.

### i Note
The metadata would be fetched from the specified client ID.

### i Note
Adding trust-store and key-store certificates isn’t supported for the on-premise system.

**Open Connectors** connection type:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region</td>
<td>Select a region from the drop-down list.</td>
</tr>
<tr>
<td>Host</td>
<td>Value is auto populated based on the selected region.</td>
</tr>
<tr>
<td>Port</td>
<td>Value is auto populated.</td>
</tr>
<tr>
<td>Organization Secret</td>
<td>Enter the organization secret. For more information on how to obtain the value, see prerequisites section.</td>
</tr>
<tr>
<td>User Secret</td>
<td>Enter the user secret. For more information on how to obtain the value, see prerequisites section.</td>
</tr>
</tbody>
</table>

**Cloud Integration** connection type:
SAP Cloud Integration

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Integration Host</td>
<td>Enter a tenant-specific address of the Cloud Integration system which can be accessed to by a dialog user (for example, an integration developer using the Web UI) or through the OData API.</td>
</tr>
<tr>
<td></td>
<td>Address of Web UI: https://&lt;Cloud Integration Host&gt;/itspaces.</td>
</tr>
<tr>
<td></td>
<td>Address of Cloud Integration OData API: https://&lt;Cloud Integration Host&gt;/api/v1.</td>
</tr>
<tr>
<td>Port</td>
<td>443 (Default value)</td>
</tr>
<tr>
<td>Authentication</td>
<td>The authentication type can be:</td>
</tr>
<tr>
<td></td>
<td>- Basic for connecting requests to server. This Basic authentication requires you to enter a username and password.</td>
</tr>
<tr>
<td></td>
<td>- OAuth2ClientCredentials for connecting requests to the server. The OAuth2ClientCredentials authentication needs you to provide:</td>
</tr>
<tr>
<td></td>
<td>- Client ID</td>
</tr>
<tr>
<td></td>
<td>- Client Secret</td>
</tr>
<tr>
<td></td>
<td>- Token URL</td>
</tr>
</tbody>
</table>

iNote
The values you provide for the above, are the ones associated with your Cloud Integration tenant.

Also see:
- Setting Up OAuth for Cloud Integration in Cloud Foundry [page 380]

6. In the CATALOG SERVICE SETTINGS section, enter the required details.

Catalog Service Settings

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Prefix</td>
<td>Specify the path prefix.</td>
</tr>
</tbody>
</table>
### Field Name | Description
---|---
Service Collection URL | Specify the relative path from where the catalog service should be fetched. For example, `/CATALOGSERVICE/ServiceCollection`.

**i Note**

This option is only relevant for SAP Gateway-enabled SAP NetWeaver systems. A catalog service retrieves a list of all available OData services on SAP Gateway. It is based on a catalog service pattern proposed by Microsoft and consists of an implementation approach of the catalog service pattern in the context of SAP Gateway. Only use the field if you want to fetch services from SAP Gateway. For more information on the catalog service, see [here](https://example.com).

Catalog URL | Automatically populates the complete URL in the format `https://<host>:<port>/path prefix/Catalog Service`.

Authentication | Select the authentication method to be used for connection requests to the server. By default, the authentication type is set to `None`. If you choose `Basic`, then provide a user name and password as authentication credentials in the respective fields.

**i Note**

Catalog service settings aren’t applicable for Open Connectors type connection and SAP Cloud Integration.

7. Save the changes.

You can view the created API Provider, by navigating to [Configure API Provider](#). Further, you can test the connection of the API Provider, by navigating to the API Provider details page and selecting [Test Connection](#).

**i Note**

The following table lists the attributes considered for Test Connection:
A fixed socket timeout of 5 seconds and connection timeout of 3 seconds is added to the HttpClient, which makes the HEAD call to the API Provider.

For more information, see https://blogs.sap.com/2017/07/27/blog-series-api-providers/.

## 1.5.1.9.1.1 Setting Up OAuth for Cloud Integration in Cloud Foundry

For Cloud Integration type Provider there are two types of authentications: Basic and OAuth2ClientCredentials.

### Context

Use Basic authentication for sending requests to server. Basic authentication requires you to enter a username and password.

Use OAuth2ClientCredentials for connecting requests to the server. This OAuth2ClientCredentials authentication requires you to enter `clientId`, `clientSecret`, and `tokenUrl`.

### Note

The values you provide for the above, are the ones associated with your Cloud Integration tenant.

Generate `clientId`, `clientSecret`, and `tokenUrl` Associated with Your Cloud Integration Tenant

### Procedure

1. In your web browser, open the SAP BTP Cockpit - https://account.hana.ondemand.com/cockpit.
2. From your Subaccount, navigate to Spaces in your Cloud Foundry environment and choose Services > Service Marketplace.
3. Choose **Process Integration Runtime** and choose **Create**.
4. In the **Create Instance** dialog that opens, choose the **api** plan.
5. In the section **Specify parameters**, paste the following JSON codes, to assign a specific role.

   ```json
   "roles": [
   "AuthGroup_IntegrationDeveloper",
   "AuthGroup_Administrator"
   ]
   ``

6. Choose **Next** until you reach the **Confirm** section
7. In the **Confirm** section, enter a unique **Instance Name** and choose **Finish**.

**Results**

The creation of service instance is successful.

**Next Steps**

Now, for the created service instance, generate a service key from the steps given below:

**Creating a Service Key**

**Procedure**

1. Choose the created service instance link from the visible list.
2. In the left-hand pane, navigate to **Service Keys** > **Create Service Key**.
3. In the **Create Service Key** dialog that opens, provide a **Name** and **Description** (optional).
4. Choose **Save**. The client credentials like **url**, **clientId**, **clientSecret**, and **tokenUrl** details appear for the given instance name.

The **clientId** and **clientSecret** are necessary credentials required to fetch the Bearer Token.

The **tokenUrl** is used to fetch the Bearer Token.

   ```json
   Example:  
   "clientid": "sb-37b5d80e-27f9-4bf58-c29f2a156df816289|it!
   b12456",  
   "clientsecret": "31lccbf6-86ef-48bb-910c-a7ee79ea29f
   $h6xDUyIUoe45qIDGGBhzK9zvDczFeePUHDLBshnMCK="",  
   "url": "https://isuitetestcpi.it-
   adev001.cfapps.sap.hana.ondemand.com",  
   "tokenurl": "https://
   isuitetestcpi.authentication.sap.hana.ondemand.com/oauth/token"
   ```

   **Copy** the client credentials in a notepad and use it while creating the API Provider.
To Discover and Create an API Proxy for Cloud Integration

Context

The service keys provide you with `clientId` and `clientSecret`. The `clientId` can be used as username and secret can be used as password if you would like to connect to your integration flows of Cloud Integration via Basic Authentication. Alternatively, you can leverage the `clientId`, `clientSecret`, and `tokenUrl` from the service keys file to get the OAuth access token and then connect to your integration flow of Cloud Integration via OAuth access token approach. For more information, see Creating an API from SAP Cloud Integration API Provider [page 401].

1.5.1.9.1.2 Principal Propagation

Principal propagation is a process that provides a secure way of forwarding the identity of a cloud user to the Cloud Connector, and from there to an on-premise system. The user information is kept confidential and, even more importantly, not changed during transit.

Prerequisites

- Your on-premise back-end system supports X.509 certification.
- On-premise connectivity is enabled in your Cloud Connector application, Cloud Controller settings. To verify the same:
  - Navigate to Cloud To On-Premise PRINCIPAL PROPAGATION in your Cloud Connector application, Cloud Controller settings.
  - Using the Name and Description column, identify the IDP connected to your Cloud Foundry subaccount, which needs to be trusted for principal propagation.
  - In the Trusted column, check if on-premise connectivity is enabled, else select the checkbox to enable it.

For information on how to connect your system via Cloud Connector, see Cloud Connector

Context

A principal forwarded to the API Proxy is validated with the IDP connected to the user’s subaccount on Cloud Foundry where is enabled. To obtain an OAuth token, which is validated, the user has to create credentials using on-premise-connectivity plan in the Cloud Foundry environment.

In API Management the Principal Propagation supports two flows namely:

- **Principal Propagation from the Neo to the Cloud Foundry Environment**: Enable an application in your subaccount in the Neo environment to access an API Management proxy created on a Cloud Foundry based API Management without a user login. For this scenario to work, the Neo subaccount needs to be trusted by the Cloud Foundry subaccount where API Management, API portal is enabled. Now, the
application on Neo can call API Management proxy using OAuth2SAMLBearer destination. For step-by-step details, see Principal Propagation from the Neo to the Cloud Foundry Environment [page 383].

- **Principal Propagation from the same Cloud Foundry subaccount**: Enable an application in your subaccount in the Cloud Foundry environment to access an API Management proxy created on the same Cloud Foundry based API Management without a user login. The JWT user token in your application can be exchanged with the API Management token using the Service Key credentials created for API Management. The application on Cloud Foundry can call API Management proxy using OAuth2UserTokenExchange destination. For step-by-step details, see Principal Propagation from the Same Cloud Foundry Subaccount [page 386].

**Related Information**

On-Premise Connectivity Plan [page 54]

### 1.5.1.9.1.2.1 Principal Propagation from the Neo to the Cloud Foundry Environment

**Prerequisites**

- You have created a Service Key by creating a service instance using the on-premise connectivity plan. For more details, see On-Premise Connectivity Plan [page 54].

**Context**

Enable an application in your subaccount in the Neo environment to access an API Management proxy created on a Cloud Foundry based API Management without a user login. For this scenario to work, the Neo subaccount needs to be trusted by the Cloud Foundry subaccount where API Management is enabled. Now, the application on Neo can call API Management proxy using OAuth2SAMLBearer destination.

**Procedure**

1. Create Trust between the Neo Subaccount and the Cloud Foundry Subaccount. For detailed steps, see Create Trust between Subaccounts.  
2. Create a Destination to the API Proxy that you want to call using principal propagation.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Technical name of the destination. It can be used later on to get an instance of that destination. It must be unique for the global account.</td>
</tr>
<tr>
<td>URL</td>
<td>Enter the API Proxy URL for the proxy you want to call.</td>
</tr>
<tr>
<td>Authentication</td>
<td>OAuth2SAMLBearerAssertion</td>
</tr>
<tr>
<td>Proxy Type</td>
<td>Internet</td>
</tr>
<tr>
<td>Audience</td>
<td>Copy the value of entityID property of the SAML 2.0 metadata representing your subaccount in the Cloud Foundry environment.</td>
</tr>
</tbody>
</table>

**Tip**

You can open the metadata of the subaccount in the Cloud Foundry environment using the token URL you obtain from the Service Key:

https://<your subaccount's subdomain>.authentication.<SAP BTP host>/saml/metadata

Example:


Example of audience/entityID:

demo.aws-live-us10

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Key</td>
<td>Enter the clientId obtained from the Service Key in the created service instance using on-premise connectivity plan.</td>
</tr>
<tr>
<td>Token Service URL</td>
<td>Enter the token url obtained from the Service Key in the created service instance using on-premise connectivity plan.</td>
</tr>
</tbody>
</table>

The token service URL is defined in the Location attribute of the element marked as AssertionConsumerService, like this:

<md:AssertionConsumerService Location="<Token Service URL>" Binding="urn:oasis:names:tc:SAML:2.0:bindings:URI" index="1"/>

**Tip**

You can open the metadata of the subaccount in the Cloud Foundry environment using the token URL you obtain from the Service Key:

Example: https://<tenant-specific-route-for-your-business-app>.sap.hana.ondemand.com/oauth/token
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token Service Password</td>
<td>Enter the <em>clientSecret</em> obtained from the Service Key in the created service instance using on-premise connectivity plan.</td>
</tr>
<tr>
<td>System User</td>
<td>Empty</td>
</tr>
<tr>
<td></td>
<td>If you have not generated the client credentials (clientId, ClientSecret, tokenUrl and application url) yet, see On-Premise Connectivity Plan [page 54]</td>
</tr>
</tbody>
</table>

For more information on creating a destination, see here.

3. Use the following sample source code to consume the above created destination in your application.

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        // Look up the connectivity configuration API
        Context ctx = new InitialContext();
        ConnectivityConfiguration configuration = (ConnectivityConfiguration) ctx.lookup("java:comp/env/connectivityConfiguration");
        // Get destination configuration
        DestinationConfiguration destConfiguration = configuration.getConfiguration("APIProxyTest"); // Name of Destination
        if (destConfiguration == null) {
            response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                    String.format("Destination %s is not found. Hint:" +
                    " Make sure to have the destination configured.", "pptest"));
            return;
        }
        AuthenticationHeaderProvider authHeaderProvider = (AuthenticationHeaderProvider) ctx.lookup("java:comp/env/myAuthHeaderProvider");
        // retrieve the authorization header for OAuth SAML Bearer principal propagation
        List<AuthenticationHeader> samlBearerHeader = authHeaderProvider.getOAuth2SAMLBearerAssertionHeaders(destConfiguration);
        LOGGER.debug("JWT token from CF XSUAA: " + samlBearerHeader.get(1).getValue());
        response.getWriter().println("JWT token from CF XSUAA: " + samlBearerHeader.get(1).getValue());
        String destinationURL = destConfiguration.getProperty("URL");
        URL url = new URL(destinationURL);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Authorization",
                samlBearerHeader.get(1).getValue());
        connection.setConnectTimeout(10000);
        connection.setReadTimeout(60000);
        int responseCode = connection.getResponseCode();
        BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        String inputLine;
        StringBuffer responseBody = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            responseBody.append(inputLine);
            System.out.println(inputLine);
            responseCode = connection.getResponseCode();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```
Principal propagation from a Neo subaccount to Cloud Foundry Subaccount is established.

### 1.5.1.9.1.2.2 Principal Propagation from the Same Cloud Foundry Subaccount

Transfer the principal (user details) within the same Cloud Foundry subaccount by exchanging tokens.

#### Prerequisites

- You have created a Service Key by creating a service instance using the on-premise connectivity plan. For more details, see On-Premise Connectivity Plan [page 54].

#### Context

When an application has to communicate with another application in the same subaccount on Cloud Foundry, exchange of token has to take place for secure passage of user details. To make this step easier, you can use the OAuth2UserTokenExchange authentication type, where the Destination service performs all these steps automatically and lets you simplify your application development. Therefore, the JWT user token in your application can be exchanged with the API Management token using the Service Key or client credentials by using OAuth2UserTokenExchange. For more details on OAuth2UserExchange destination, see here.

#### Procedure

1. Create a Destination to the API Proxy that you want to call using principal propagation. Enter the following details while configuring a destination of type OAuth2UserTokenExchange. Also, if you have not generated the client credentials (clientId, ClientSecret, tokenUrl and application url) yet, see On-Premise Connectivity Plan [page 54].

```
Example:
Type=HTTP
clientId=Client id from the credentials
Authentication=OAuth2UserTokenExchange
Name= Name of the destination
tokenServiceURL= Token url from the credentials
ProxyType=Internet
URL=API Proxy URL for the proxy to be called
tokenServiceURLType=Dedicated
clientSecret=<Client Secret from the credentials>
```
2. Exchange the JWT user token in the application with the API Management, API, portal application via OAuthUserTokenExchange authentication type for HTTP destinations. For more detailed information, see here.

Next Step:
Use the token received from the OAuthUserTokenExchange to call an API Proxy and consume the above-created destination from your application.

1.5.1.9.1.2.3 Principal Propagation between two different Subaccounts in Cloud Foundry Environment

Propagate the identity of a user between two Cloud Foundry applications that are located in different subaccounts or regions.

Prerequisites

- You have two applications (application 1 and application 2) deployed in Cloud Foundry spaces in different subaccounts in the same region or in different regions.
- You have an instance of the Destination service bound to application 1.
- You have a user JWT (JSON Web Token) in application 1 where the call to application 2 is performed.

Procedure

Assemble IdP Metadata for Subaccount 1

1. From Subaccount 1 Destinations Download Trust, download the X.509 certificate of subaccount 1. The content of the file is shown as:

```
-----BEGIN CERTIFICATE-----<content>-----END CERTIFICATE-----
```

Below, we refer to the value of <content> as `$S1_CERTIFICATE`.

2. In the cockpit, navigate to the overview page of subaccount 1. Here you can see the landscape domain, subaccount ID and subdomain. Below, we refer to the landscape domain as `$S1_LANDSCAPE_DOMAIN`, to the subaccount ID as `$S1_SUBACCOUNT_ID` and to the subdomain as `$S1_SUBDOMAIN`.

3. In your browser, call https://$S1_SUBDOMAIN.authentication.$S1_LANDSCAPE_DOMAIN/saml/idp/metadata and download the XML file. Within the XML file you can find the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" ID="<id>" entityID="<alias>">
...
```

Sample Code
Below, we refer to the value of <alias> as \(S1\_ALIAS\).

4. Assemble the new IdP metadata for subaccount 1 by replacing the \(S\{\ldots\}\) placeholders in the following template with the values determined in the previous steps:

```xml
<ns3:EntityDescriptor
    ID="cfapps.\$(S1\_LANDSCAPE\_HOST)/\$(S1\_SUBACCOUNT\_ID)"
    entityID="cfapps.\$(S1\_LANDSCAPE\_HOST)/\$(S1\_SUBACCOUNT\_ID)"
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:ns2="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ns4="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:ns3="urn:oasis:names:tc:SAML:2.0:metadata">
    <ns3:SPSSODescriptor AuthnRequestsSigned="true"
        protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <ns3:KeyDescriptor use="signing">
            <KeyInfo>
                <KeyName>\$(S1\_ALIAS)</KeyName>
                <X509Data>
                    <X509Certificate>
                        \$(S1\_CERTIFICATE)
                    </X509Certificate>
                </X509Data>
            </KeyInfo>
        </ns3:KeyDescriptor>
    </ns3:SPSSODescriptor>
    <ns3:IDPSSODescriptor
        WantAuthnRequestsSigned="true"
        protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <ns3:KeyDescriptor use="signing">
            <KeyInfo>
                <KeyName>\$(S1\_ALIAS)</KeyName>
                <X509Data>
                    <X509Certificate>
                        \$(S1\_CERTIFICATE)
                    </X509Certificate>
                </X509Data>
            </KeyInfo>
        </ns3:KeyDescriptor>
    </ns3:IDPSSODescriptor>
</ns3:EntityDescriptor>
```

Establish Trust between Subaccount 1 and Subaccount 2

1. In the cockpit, navigate to the overview page for subaccount 2.
2. From the left panel, select Security > Trust Configuration >. Choose New Trust Configuration.
3. Paste the assembled IdP metadata for subaccount 1 in the Metadata text box and uncheck Available for User Logon.
5. Enter a <Name> for the trust configuration and choose Save.

**Note**

Additionally, you must add users to this new trust configuration and assign appropriate scopes to them.
Create an OAuthSAMLBearerAssertion Destination for Application 1

1. In the cockpit, navigate to the overview page for subaccount 2.
2. Here you can see the landscape domain, subaccount ID and subdomain of subaccount 2. Below, we refer to the landscape domain as \( \$\{S2\_LANDSCAPE\_DOMAIN\} \), to the subaccount ID as \( \$\{S2\_SUBACCOUNT\_ID\} \) and to the subdomain as \( \$\{S2\_SUBDOMAIN\} \).
3. In your browser, call \( \text{https://}\$\{S2\_SUBDOMAIN\}.\text{authentication.}\$\{S2\_LANDSCAPE\_DOMAIN\}/saml/idp/metadata \) and download the XML file. Within the XML file, you can find the following structure.

### Sample Code

```xml
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
  ID="" entityID="" alias="">
  ...
</md:EntityDescriptor>
<md:AssertionConsumerService
  Binding="urn:oasis:names:tc:SAML:2.0:bindings:URI"
  Location="token" index=""/>
</md:EntityDescriptor>
```

Below, we refer to the value of \(<\text{alias}>\) as \( \$\{S2\_ALIAS\} \).
4. In cockpit, navigate to subaccount 1.
5. From the left panel, select Connectivity » Destinations »
6. Choose New Destination and configure the values as described below. Replace the \( \$\{\ldots\} \) placeholders with the values you determined in the previous steps and sections.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Choose any name for your destination. You will use this name to request the destination from the Destination service.</td>
</tr>
<tr>
<td>Type</td>
<td>HTTP</td>
</tr>
<tr>
<td>URL</td>
<td>The URL of application 2, identifying the resource you want to consume.</td>
</tr>
<tr>
<td>Proxy Type</td>
<td>Internet</td>
</tr>
<tr>
<td>Authentication</td>
<td>OAuth2SAMLBearerAssertion</td>
</tr>
<tr>
<td>Audience</td>
<td>entityId</td>
</tr>
<tr>
<td>Client Key</td>
<td>The clientid of the XSUAA instance in subaccount 2. Can be acquired via a binding or service key.</td>
</tr>
<tr>
<td>Token Service URL</td>
<td>The URL of the XSUAA instance in subaccount 2. Can be acquired from (&lt;\text{token}&gt;).</td>
</tr>
<tr>
<td>Token Service URL Type</td>
<td>Dedicated</td>
</tr>
<tr>
<td>Token Service User</td>
<td>The clientid of the XSUAA instance in subaccount 2. Can be acquired via a binding or service key.</td>
</tr>
<tr>
<td>Token Service Password</td>
<td>The clientsecret of the XSUAA instance in subaccount 2. Can be acquired via a binding or service key.</td>
</tr>
</tbody>
</table>

Additional Properties
Choose Save.

**Consume the Destination and Execute the Scenario**

To perform the scenario and execute the request from application 1, targeting application 2, proceed as follows:

1. Decide on where the user identity will be located when calling the Destination service. For details, see [User Propagation via SAML 2.0 Bearer Assertion Flow](#). This will determine how exactly you will perform step 2.
2. Execute a “find destination” request from application 1 to the Destination service. For details, see [Consuming the Destination Service](#) and the REST API documentation.
3. From the Destination service response, extract the access token and URL, and construct your request to application 2. See "Find Destination" Response Structure for details on the structure of the response from the Destination service.

### 1.5.1.10 Create an API Proxy

Create API proxies.

When you create an API, you can choose from the following options:

- Import an API: If you have an API definition, you can reuse it by importing it into the API Management. For more information, see [Import an API](#).
- Create from API Management: Create an API proxy from scratch by defining the resources, documentation, and by attaching policies. For more information, see [Create an API proxy](#).

**Note**

- API Management supports OData, REST, and SOAP services.

An API Proxy consists of a virtual host and a base path. The base path can be identical for multiple API proxies, provided API proxies have different virtual hosts. This means, for an API Proxy, the combination of the virtual host and base path should be unique.

The example below explains the same, where API1 is proxy 1, AP2 is proxy 2, VH1 is Virtual Host 1, VH2 is the Virtual Host 2, and BP(A) is the base path.

Example: AP1 = VH1 + BP(A)

AP2 = VH2 + BP(A)
1.5.1.10.1 API Versioning

Versioning allows the creation and management of multiple releases of an API. You can version an API proxy when you want to improve, upgrade, or customize the functional behavior provided by a currently existing API proxy.

Prerequisites

You need the APIPortal.Administrator role to use the versioning feature.

A new version of the API proxy is created when:

- Incompatible changes, such as structural changes or changes in payload, are planned.
- Additional policies are enforced, and a step-by-step client transition is needed.
- API developer wants to model changes, which will be exposed as new version without affecting the consumption of existing API version.
- There’s change in the billing model.

You can add a version to an API, when you’re creating the API proxy. This applies to creation from:

- API portal
- Import of API definition file to the API portal
- Creation from API designer
- Creation from APIs available from the SAP API Business Hub

Multiple versions of the API, can coexist at both runtime and design time.

The pattern for the name and base path for an API Proxy, when you’re using the version attribute, and have chosen, for example, v1 as the version is:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>v1</td>
</tr>
<tr>
<td>Name</td>
<td>Name_v1</td>
</tr>
<tr>
<td>Base Path</td>
<td>/v1/SalesOrder</td>
</tr>
</tbody>
</table>

In case you don’t provide name and base path in this required pattern, the system appends the version to the name, and prepends the version to the base path, to create a unique version of the API proxy.

For the version, we recommend that you use alphanumeric values. Based on the version you’ve provided, the system appends the version value to the API proxy name and base path, creating a unique version.

→ Remember

Once you’ve versioned an API Proxy, you can’t edit the version or the base path.
1.5.10.11 Creating a Versioned API

Creating a versioned API Proxy from a deployed, versioned, or nonversioned API Proxy in the API Management, API Portal.

Context

You can create a versioned API Proxy, from either a previously versioned API or a nonversioned API that have been deployed, from the API Portal. To know more about creating a versioned API Proxy when creating it from scratch in the API Portal, see API Versioning [page 391].

Procedure

1. Log on to the API portal.

2. From the navigation bar, choose (Develop).

   A list of registered APIs appears in the catalog.

3. Select the API from which you want to create a new versioned API. This can be either already versioned API, or a nonversioned API.

4. From the top-right corner of the Overview screen, choose ... New Version.

5. In the Create New Version screen, in the Version field, enter a new version and choose Create. You can’t edit the name and the base path. However, once you enter a new version, the version is appended to the name and prepended to the base path to create a unique API Proxy version.

   If the version you enter is v1, the name will be Name_v1, and the basepath will /v1/SalesOrder.

6. You can then choose to Save or Deploy the new version of this API Proxy.
1.5.1.10.2 API Proxy States

As an API Management administrator, you can set states for an API proxy while creating or updating the API proxy.

The following states can be set for an API proxy:

<table>
<thead>
<tr>
<th>State</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>A version of an API meant for exploratory purposes.</td>
</tr>
<tr>
<td>Beta</td>
<td>A version of an API that isn’t meant for productive use.</td>
</tr>
<tr>
<td>Active</td>
<td>A version of an API that is meant for productive use.</td>
</tr>
<tr>
<td>Deprecated</td>
<td>When an API is marked as deprecated, the customer is encouraged to use a successor API.</td>
</tr>
<tr>
<td>Decommissioned</td>
<td>A version of an API where the service is no longer available, and can’t be used productively.</td>
</tr>
</tbody>
</table>

When you deprecate or decommission an API proxy, you must provide a deprecation or decommissioning date, as well as a successor API. This could be another API within the API portal or an external link. You can only mark an API proxy as **Deprecated** or **Decommissioned** while updating the proxy and not while creating it.

The API state is to be used only for demarcation, and doesn’t play a role in the governance or lifecycle of the API.

![iNote]

An application developer can view the states of the deprecated and decommissioned API proxies (if published in the API portal) in the API details screen of the API business hub enterprise.

1.5.1.10.3 Create an API

This topic describes the steps to create an API from the API portal.

**Prerequisites**

To build APIs, you must do the following:

- Make sure that you have the REST, OData, or SOAP URL of the service that you want to expose as an API.
- Browse for a service on a specific API provider. To do so, you must configure the required API provider on the **Configure** tab.
- You’ve created an instance against ORG/USER secret for creating an API Proxy from an Open Connector type API Provider.
Context

Instead of consuming services directly, application developers can access APIs exposed via API Management. You do so, by creating an API proxy that camouflages the service you want to expose. The API maps a publicly available HTTP endpoint to back-end services. Creating this API proxy lets API Management handle the security and authorizations required to protect, analyze, and monitor your services.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
   
   A list of registered APIs appears in the catalog.
   
   Alternatively, browse for a service on a specific API provider. To do so, you must configure the required API provider on the Configure tab. You can view the number of calls made for an API in the current month. The data is visible for each API in the Calls column and also on the details screen of the individual API.
   
   You can click the refresh icon to obtain the latest data.

<table>
<thead>
<tr>
<th>i Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>There’s some delay in reflecting the latest data.</td>
</tr>
</tbody>
</table>

   The notion used to display the data follows metric specifications, for example:
   ○ 999 shows as 999 and 1000 shows as 1k
   ○ 999000 shows as 999 K and 1000000 shows as 1M
   ○ 1500000 shows as 1.5M and 1000000000 shows as 1G

3. To expose a service as an API, choose [Create].
4. If you want to browse for an OData service for a provider that you’ve already configured, proceed as follows:
   a. Select the API Provider radio button.
   b. Select the required open connector type provider from the API Provider dropdown list.
      
      The dropdown list contains the providers that you’re connected to. If the provider you need isn’t listed here, add it on the Configure tab.
   c. To view the list of OData services available in the provider, choose Discover.
   d. Select the required service.
   e. If you want to link the API to the target server, select the checkbox Link API Provider.
      
      If you deselect the Link API Provider checkbox, the API proxy is no longer linked to any API provider. It now acts just like a URL-based API. However, since the API created is originally of type OData, you can’t add or delete resources to it.
      
      Also, since such APIs aren’t linked to any API provider, you can’t use the Synchronize option to update the API with the latest version that might be available in the backend.
   f. Select a virtual host alias from the Host Alias dropdown.
      
      The details of the API name, description, API base path, and service type are automatically populated.
g. Optional: Enter a version for your API proxy.

When you choose to version your API proxy, its name is appended with the version, and its basepath are prepended with the version. For example, if the version you enter is v1, the name is Name_v1, and the basepath is /v1/SalesOrder. For more information, see API Versioning [page 391]

h. Choose Create.

i. If you want to add SAP documentation annotations to the API documentation, choose Yes for Documentation.

### i Note
This field is only displayed if you’re fetching services from SAP Gateway Systems. For more information about SAP documentation annotations, see Extended Support of Long Texts in the Metadata.

5. If you want to create a proxy for an Open Connector instance for a provider that you’ve already configured, proceed as follows:

a. Select the API Provider radio button.

b. Select the required provider from the API Provider dropdown list.

The dropdown list contains the providers that you’re connected to. If the provider you need isn’t listed here, add it on the Configure tab.

c. To view the list of instances available in the provider, choose Discover.

d. Select the required instance.

e. Link the API to the target server, and select the checkbox Link API Provider.

If you deselect the Link API Provider checkbox, the API proxy is no longer linked to any API provider. It now acts just like a URL-based API. However, since the API created is originally of type OData, you can’t add or delete resources to it.

Also, since such APIs aren’t linked to any API provider, you can’t use the Synchronize option to update the API with the latest version that might be available in the backend.

f. Select a virtual host alias from the Host Alias dropdown.

The details of the API name, description, API base path, and service type are automatically populated.

### i Note
Name and basepath shouldn’t contain spaces.

g. Optional: Enter a version for your API proxy.

When you choose to version your API proxy, its name is appended with the version, and its basepath is prepended with the version. For example, if the version you enter is v1, the name is Name_v1, and the basepath is /v1/SalesOrder. For more information, see API Versioning [page 391]

h. Choose Create.

On creating the proxy, an encrypted key value map is created with the following name apim.oc.instance.token and key name default. Also, an open connector policy is attached to the incoming POST flow request of the target endpoint of the APIProxy.

### i Note
- For an API Proxy, you can have only one open connector policy attached and the content of the open connector policy can’t be modified.
An API Proxy consists of a virtual host and a base path. The base path can be identical for multiple API proxies, provided the API proxies have different virtual hosts. This means, for an API Proxy, the combination of the virtual host and base path should be unique.

The example below explains the same, where AP1 is proxy 1, AP2 is proxy 2, VH1 is Virtual Host 1, VH2 is the Virtual Host 2, and BP(A) is the base path.

Example:
AP1 = VH1 + BP(A)
AP2 = VH2 + BP(A)

On deleting the proxy, the encrypted key value map created above is also deleted. Further, while exporting the API, encrypted key value map created above isn’t exported with the API.

6. If you want to browse for an existing API proxy, proceed as follows:
   a. Select the API Proxy radio button.
      To view all available API proxies, choose Discover. Select the required API proxy from the dropdown list.
   b. Enter a name and description for the API.
   c. Optional: Enter a version for your API Proxy.
      When you choose to version your API Proxy, its name is appended with the version, and its basepath is prepended with the version. For example, if the version you enter is v1, the name is Name_v1, and the basepath is /v1/SalesOrder. For more information, see API Versioning [page 391]
   d. Select a virtual host alias from the Host Alias dropdown.
   e. In the API Base Path field, provide a path prefix for the API.

7. If you have the URL of the OData service, proceed as follows:
   a. Select the URL radio button.
   b. Enter the OData service URL in the URL field. For example, http://<host>:<port>/SFlight.

   i Note
   Ensure that the service URL you provide doesn’t redirect to a different URL. That is, check if the service URL you’re trying to access is temporarily or permanently moved to a different location. If it does so, then it’s recommended that you provide the new location (redirected URL, if exists) of the service.
   For more information about how to handle URL redirection, see Handling URL Redirects in an API Proxy Using Policies [page 368].
   c. Enter a name and description for the API.
   d. Optional: Enter a version for your API Proxy.
      When you choose to version your API Proxy, its name is appended with the version, and its basepath is prepended with the version. For example, if the version you enter is v1, the name is Name_v1, and the basepath is /v1/SalesOrder. For more information, see API Versioning [page 391]
   e. Select a virtual host alias from the Host Alias dropdown.
   f. In the API Base Path field, provide a path prefix for the API. For example, v1/SFlight.
   g. In the Service Type field, enter OData.

8. If you have the URL of the REST service, proceed as follows:
   a. Select the URL radio button.
   b. Enter the REST-based service URL in the URL field. For example, http://<host>:<port>/SFlight.
Note

Ensure that the service URL you provide doesn’t redirect to a different URL. That is, check if the service URL you’re trying to access is temporarily or permanently moved to a different location. If it does so, then it’s recommended that you provide the new location (redirected URL, if exists) of the service.

For more information about how to handle URL redirection, see Handling URL Redirects in an API Proxy Using Policies [page 368].

c. Enter a name and description for the API.
d. Optional: Enter a version for your API Proxy.

When you choose to version your API Proxy, its name is appended with the version, and its basepath is prepended with the version. For example, if the version you enter is v1, the name is Name_v1, and the basepath is /v1/SalesOrder. For more information, see API Versioning [page 391]
e. Select a virtual host alias from the Host Alias dropdown.
f. In the API Base Path field, provide a path prefix for the API. For example, v1/SFlight.
g. In the Service Type field, enter REST.

9. If you have the URL of the SOAP service, proceed as follows:
   a. Choose the URL radio button.
b. Enter the SOAP service URL in the URL field. For example, http://<host>:<port>/SFlight.
c. Enter a name and description for the API.
d. Optional: Enter a version for your API Proxy.

   When you choose to version your API Proxy, its name is appended with the version, and its basepath is prepended with the version. For example, if the version you enter is v1, the name is Name_v1, and the basepath is /v1/SalesOrder. For more information, see API Versioning [page 391]
e. Select a virtual host alias from the Host Alias dropdown.
f. In the API Base Path field, provide a path prefix for the API. For example, v1/SFlight.
g. In the Service Type field, choose SOAP.

10. Choose Create.

Note

For an OData service, all the associated artifacts appear on the different tab pages mentioned below. The Resources tab lists all the resources associated with the API. The API documentation with SAP documentation annotations, if selected, is also fetched from the metadata.

For a SOAP- and REST-based service, the Resources tab appears. Add the resources manually.

Note

In case you want to restrict your users from accessing all the resources associated with the API Proxy, you need to create a new Product, add the required API to the Product, and select the resources for
11. For a REST service, add a resource as follows:
   a. Choose + (Add).
   b. In the popup, enter a title and path prefix for the resource.
   c. Select the methods that need to be supported for this resource.
   d. Add descriptions in the editor and choose Add.
      The added resource appears on the Resources tab.
   e. Choose Add to add more resources to the same API.
   f. Proceed to Step 14.

12. For a SOAP service, add a resource as follows:
   a. Choose + (Add).
   b. Enter a title and specify the SOAP operation name in the path prefix.
   c. Use the editor to enter the relevant API documentation in the description field, and choose Add.
      The added resource appears on the Resources tab. By default, only the POST operation is selected.
   d. Choose Add to add more resources to the same API.
   e. Proceed to Step 14.

13. For an OData service, select the methods that the application developer can perform:
   - Get: Read an entity.
   - Post: Create an entity.
   - Put: Update an entity.

   **Note**

   API Management generates PUT operation as the default update operation. However, you can override the default update operation for API Proxy of type OData. For more information, see Overriding the Default Update Operation for API Proxy of Type OData [page 403].
○ **Delete**: Delete an entity.

**i Note**

Only the supported methods for each resource appear on the UI. By default, only permitted methods are selected.

14. For a given resource, choose *Show/Hide* to view the list of properties and their associated API documentation. You can add descriptions for each resource in the editor.

**i Note**

For a given resource, choose *Open API Designer* and correct the errors in swagger definition, if any. The error message displayed on the screen helps in error detection and correction. Choose *Save* after making the necessary corrections in the swagger file.

See the example in the following screen:

15. To define policies on the API, go to the *Policies* tab. For more information about how to create a policy, see [Create a Policy](page 451).

16. If you want to define multiple proxy endpoints, navigate to *Proxy EndPoint* tab.

   In the *Proxy Endpoint Properties* section, choose *Add*. Enter the *Property Name* and the *Values*. For the Proxy Endpoint property specifications, see [Proxy Endpoint Properties](page 355).

17. If you want to define multiple route rules, navigate to *Proxy EndPoint* tab. In the *Route Rules* section, choose *Add*.

**i Note**

When the API is created, the default route rule is set. It points to the default target endpoint and no rule is attached to it. Use the option *None* to ensure that no request is routed to any target endpoint. If there are multiple route rules, the rules are evaluated in sequence as displayed on the screen.

For more information on how to define multiple target endpoints using Route Rule, see [Enable Dynamic Routing](page 360).

18. If you want to define multiple target endpoints, navigate to *Target EndPoint* tab. In the *Target Endpoint Properties* section, choose *Add*.
In the **Target Endpoint Properties** section, choose **Add**. Enter the **Property Name** and the **Values**. For the Target Endpoint property specifications, see *Target Endpoint Properties [page 356]*.

19. Once you’ve filled in all the required details of the API, you can select one of the following 2 actions for the API:

<table>
<thead>
<tr>
<th>Action</th>
<th>Resulting API State</th>
<th>Future Action on API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>Not Deployed</td>
<td>Deploy</td>
</tr>
<tr>
<td></td>
<td>API is available only in the API portal, and isn’t available for product assignments.</td>
<td>API is deployed and is ready for product assignments.</td>
</tr>
<tr>
<td>Deploy</td>
<td>Deployed</td>
<td>Undeploy</td>
</tr>
<tr>
<td></td>
<td>Only deployed APIs can be selected for product publishing.</td>
<td>If any API is undeployed after being published, it’s removed from the developer portal. When the API is deployed again, the product is updated. You can bring down an API without having to delete it from the product assignment. You can’t undeploy an API if it’s the only one associated with the product.</td>
</tr>
</tbody>
</table>

20. Once you’ve created an API, you can do the following:

- Go to the **Resources** tab, where you can test the API resource by using the **Operations** button beside it. For more information, see *Test APIs [page 464]*.
- Go to more options and from the drop-down select **Synchronize**. This functionality updates the API with the latest version available in the back end.

**Note**
- This functionality is available only for OData API proxies.
- The discovered OData API proxies, which already exist and have their resource documentation in OAS 2.0 are rendered in OAS 2.0 after synchronization.
- With OAS 3.0 support, the discovered OData API proxies, which have their resource documentation in OAS 3.0 are rendered in OAS 3.0 after synchronization.

**Related Information**

Create an API from API Designer [page 405]
1.5.10.3.1 Creating an API from SAP Cloud Integration API Provider

Create an API proxy from list of APIs that is deployed in a SAP Cloud Integration tenant.

Prerequisites

- You should have already created an API provider of type SAP Cloud Integration Configure tab in the API portal. To do so, see Create an API Provider [page 374].

Context

Instead of consuming the API services directly, application developers can access APIs exposed via API Management. You do so, by creating an API proxy that covers the service you want to expose. The API maps a publicly available HTTP endpoint backend service. Creating this API proxy lets you handle the security and authorizations required to protect, analyze, and monitor your services. Here, you can see how to create an API proxy using an Integration Flow or an API from the list of artifacts deployed in SAP Cloud Integration tenant.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. To expose a service as an API, choose Create.
4. In order to choose an API of the type OData, REST, or SOAP and create an API Proxy, proceed as follows:
   a. In the Create API dialog, select the API Provider radio button.
   b. From the API Provider dropdown, select an API Provider that belongs to the type Cloud Integration.
      The dropdown list contains the providers that you’re connected to. If the provider you need isn’t listed, add an API provider of type Cloud Integration from the Configure tab. For more information, see Create an API Provider.
   c. Choose Discover. A list of Integration Flows and APIs belonging to type OData, REST, or SOAP appears from Cloud Integration.
   d. Choose an API to connect with the API Proxy.
   e. Choose Next.
   f. In the Authentication dialog, if you select:
      - Basic - A Basic authentication is a mode of user verification. If your selected API supports Basic authentication, enter the credentials, either Username and Password or Client ID and Client Secret. For more information, see Setting Up Basic Inbound Authentication with ClientId and Clientsecret from Service Key in the Cloud Foundry Environment.
Changing the user credentials for the API might affect proxy execution. However, if necessary, you can modify the user credentials after the proxy creation by following the instructions in the note of step (6):

- **Client Certificate** - A Client Certificate authentication is a mode of user verification. If your selected API supports Client Certificate authentication, you can upload the certificate (consisting of public and private key including the certificate chain) in the provided section. The uploaded certificate is stored under a Store (collection of certificates). For more information, see Setting Up Inbound Client Certificate Authentication, Cloud Foundry Environment.

  If you choose:
  - **Existing Store**: The Existing Store consists of an already created certificate from the Store, that is, selecting Existing Store enlists only the Stores created by uploading certificates of format .p12 or .pfx. When you choose such a certificate from the Store Name, you need to have the password for the Store handy.
    1. From the Store Name dropdown, choose an existing Store.
    2. In the section Password, type in the respective password for the existing Store.
  - **New Store**: 1. Upload a certificate of format .p12 or .pfx format by choosing Browse. For more information on the certificates, see here.
    2. Provide a unique name to the store, in the section Store Name.
    3. Provide a unique certificate name, in the field Name.
    4. Set a password for your certificate in the section Password.
    5. Choose Done.
  - **OAuth2ClientCredentials** You can choose to use OAuth2ClientCredentials authentication for your API deployed in SAP Cloud Integration. You would need to enter the following:
    - **Client ID**
    - **Client Secret**
    - **Token URL**
    The Client ID, Client Secret, and Token URL, are those obtained when you configure OAuth authentication, in particular the Client Credentials Grant variant, for Inbound calls from sender systems to the integration platform. For more information, see Setting Up OAuth Inbound Authentication with Client Credentials Grant

  In the Create API dialog, the API Details consisting of Name, Title, Description, Host Alias, API Base Path, and Service Type are auto populated.

  g. Choose Create.

  An API Provider is auto-created with the name that is populated in the Name field of the Create API dialog. This auto-created API provider helps in storing the user credentials provided in the Authentication dialog and connects to the API Proxy.

  5. A Create API screen opens with the API Proxy name on the top. You can edit the API Proxy details, if necessary and choose Save.

**i Note**

While creating API proxies for SOAP and REST, API resources aren’t autogenerated; you must add them manually.
While creating API proxies for OData API, autogeneration of resources may be possible in some cases.

6. In the top-right corner of the screen, you can either choose **Deploy** to create and deploy the API Proxy or **Cancel** to delete the API Proxy.

**Note**

Choosing **Save** will only create an API Proxy and doesn’t deploy an API Proxy. You can modify the credentials after the API Proxy execution by following the steps:

1. Navigate to the auto-created API Provider with the same name as that of your selected API in the **Develop** tab.
2. Choose **Target EndPoint**.
3. Choose the link under the **API Provider** section. This API Provider has the user credentials stored.
4. Choose **Connection**.
5. Select **Edit** on the top-right corner of the screen. Modify the user credentials.
6. Choose **Save**.

**Related Links:**
- Concepts of Secure Communication

### 1.5.1.10.3.2 Overriding the Default Update Operation for API Proxy of Type OData

When discovering an API via OData API provider, the update operation is generated automatically based on the backend OData version.

For OData V2 backend service, API Management generates PUT operation as the default update operation. For OData V4 backend service, API Management generates PATCH operation as the default update operation. However, you can override this by adding the key through the API designer for swagger 2.0 or Open API 3.0:

YAML Format:

```
x-sap-default-update-operation: put
x-sap-default-update-operation: patch
```

JSON Format:

```
"x-sap-default-update-operation": "put"
"x-sap-default-update-operation": "patch"
```

In case both the update operations are required, you can pass both the operations as an array as shown below:

YAML Format:

```
x-sap-default-update-operation: put
x-sap-default-update-operation: patch
```

JSON Format:

```
"x-sap-default-update-operation": ["put", "patch"]
```
**iNote**

Use array if only both the operations are required. You can use the API designer to add this key in the Open API specification under “info.”

```json
1 openapi: 3.0.0
2 * info:
3   title: CATALOGSERVICE
4   version: '1'
5   description: 'This service is located at [http://localhost/api/](http://localhost/api/)
6     x-sap-default-update-operation:
7       - put
8       - patch
9   servers:
10     url: 'https://155.56.129.175:443/CATALOGSERVICE_Takt_Demo'
11 tags: 
```

Once you have made the changes in the Open API specification, ensure that you synchronize the API proxy for the changes to reflect on the Resource operations.

**Related Information**

Create an API [page 393]

---

**1.5.10.4 Deploy an API**

After an API is created, you must deploy the API to make it ready for product assignments.

**Procedure**

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. To expose a service as an API, choose Create. See Create an API [page 393] for the steps on how to create an API.

   Once you've filled in all the required details of the API, refer to step 19 in Create an API [page 393] to Deploy the API.
i Note

After deploying the API proxy, there can be instances where the policies or the base path added to the API proxy are modified and saved within the API proxy. In such scenarios, the API proxy gets automatically deployed, and the changes start reflecting over the internet.

Results

Once an API is deployed, it’s available on the internet. The API can be called using the virtual host and the base path and can also be packaged as API Products.

1.5.1.10.5 Create an API from API Designer

Model APIs in the Open API format that is available on the API portal.

Prerequisites

The APIPortalAdministrator role should be assigned to you.

To build APIs, you must have the REST, OData, or SOAP URL of the service that you want to expose as an API.

Context

The API designer editor includes rich inbuilt capabilities that enable you to do the following:

- Import existing open APIs
- Download APIs
- Generate equivalent HTML output views
- Save APIs
- Validate open API syntax

The API designer supports OAS 2.0 and OAS 3.0 versions.

To know more about OAS 3.0 support in API Management, see OpenAPI Specification 3.0 [page 82].

Procedure

1. Log on to the API portal.
2. From the navigation bar, choose **Develop**.
   A list of registered APIs appears in the catalog.

3. To model an API in Open API format, choose **Create in API Designer**.

   - **Note**
     - You can now create your API in the embedded API designer
     - To update an existing API, select the API and choose **Edit in API Designer**.

4. Enter the API-related information and choose **Save**. For information on how to model the API, see **Creating APIs [page 406]**.

5. You can choose to version your API, by selecting **Save** > **Save as Version**, and entering a new value in the **Version** field. To know more about versioning your API, see **API Versioning [page 391]**.
   When you choose to version your API Proxy, it’s name will be appended with the version, and it’s basepath will be prepended with the version. For example, If the version you enter is v1, the name will be Name_v1, and the basepath will be /v1/SalesOrder.

### Related Information

- **Create an API [page 393]**

### 1.5.1.10.5.1 Creating APIs

You can create APIs in API designer.

Use the [OpenAPI Specification](#) to create APIs in API designer.

API specification can be written in YAML or JSON. In this guide, we use both YAML and JSON examples.

The API designer has a preview pane. You can write API specification in YAML in the right-hand pane and preview the corresponding reference documentation in the left-hand pane.

The following image illustrates the process of API creation in API designer.

- **Note**
  - All the API code samples shown in this document refers to only API 2.0 specification.

- **Note**
  - The image contains links to more information. You can hover over each shape for a description and click on it for more information.
Define Metadata
Define Root URL
Add Attributes
Define Operations
Define Parameters
Define Responses
Add Definitions
Add Security Definitions
Add Tags
Add External Links

- #unique_233/unique_233_Connect_42_subsection-im1 [page 408]
- #unique_233/unique_233_Connect_42_subsection-im2 [page 410]
- #unique_233/unique_233_Connect_42_subsection-im3 [page 411]
- #unique_233/unique_233_Connect_42_subsection-im4 [page 411]
- #unique_233/unique_233_Connect_42_subsection-im5 [page 412]
- #unique_233/unique_233_Connect_42_subsection-im6 [page 413]
- #unique_233/unique_233_Connect_42_subsection-im7 [page 414]
- #unique_233/unique_233_Connect_42_subsection-im8 [page 414]
- #unique_233/unique_233_Connect_42_subsection-im9 [page 414]
- #unique_233/unique_233_Connect_42_subsection-im10 [page 415]

Hover over each shape for a description. Click the shape for more information.
Define Metadata

Enter the general information or the metadata of the API:

- **swagger** - The version of the OpenAPI Specification. 2.0 being the latest version.
- **title** - A short summary of the API overall functionality.
- **description** -
  An expanded description of the API overall functionality and its usage specifics, if any.
  Descriptions can be entered in multiline. Basic html tags with appropriate attributes, can be used for styling.
  Tags that can be used are:
  - `<a>`
  - `<b>`
  - `<blockquote>`
  - `<br>`
  - `<cite>`
  - `<code>`
  - `<dd>`
  - `<dl>`
  - `<dt>`
  - `<em>`
  - `<i>`
  - `<li>`
  - `<ol>`
  - `<p>`
  - `<pre>`
  - `<q>`
  - `<small>`
  - `<span>`
  - `<strong>`
  - `<sub>`
  - `<sup>`
  - `<u>`
  - `<ul>`
  - `<img>`
  Any other tags will be auto-removed.
  For the image, only Base64 encoded content is supported.
- **version** - The version of the API.

---

**Sample Code**

API specification in YAML

```yaml
swagger: '2.0'
info:
  title: SAP Workflow API
  description: |
    This API provides functionality to work with SAP Workflow Service.
    You can, for example, start new workflow instances
```
and work with tasks.

version: 1.0

### Sample Code

#### API specification in JSON

```json
{
  "swagger": "2.0",
  "info": {
    "title": "SAP Workflow API",
    "description": "This API provides functionality to work with SAP Workflow Service.",
    "version": 1
  }
}
```

#### Sample Code

#### API specification in YAML

```yaml
swagger: '2.0'
info:
  title: SAP e-commerce API
  description: |
    This API provides functionality to build an e-commerce site selling electronic products.
    The following scenarios are covered:
    * Customer can order products
    * Customer can provide product reviews
    * Retailer can create sales orders
    * Retailer can update product stock
  version: 1.0
```

### Sample Code

#### API specification in JSON

```json
{
  "swagger": "2.0",
  "info": {
    "title": "SAP e-commerce API",
    "description": "This API provides functionality to build an e-commerce site selling electronic products.",
    "version": 1
  }
}
```

It is also a good practice to provide the license and the contacts details of the API. In the `license` object, you can provide a link to the licensing rules for the API, and in the `contact` object, you can provide details of the API provider.

#### Sample Code

#### API specification in YAML

```yaml
license:
  name: Apache-2.0
```
Define Root URL

APIs have a root URL to which the paths or the endpoints are appended. The root URL is defined by `host`, `schemes`, and `basePath` on the root level of the OpenAPI Specification:

- **host** - The host (name or IP address) serving the API. The host might include a port number. If a value for host is missing, then the host (including the port) providing the documentation is assumed also to provide the API.
- **schemes** - The transfer protocol expected by the API. Values must be either **http** or **https**. If a value for schemes is missing, then the scheme used to access the OpenAPI Specification definition becomes the scheme used to access the API.
- **basePath** - The base path on which the API is served, relative to the host. If this value is missing, then the API must be available directly under the host. The value must start with a leading forward slash (`/`).

### Sample Code

**API specification in JSON**

```json
{
  "license": {
    "name": "Apache-2.0",
    "url": "https://github.com/SAP/master/LICENSE"
  },
  "contact": {
    "name": "SAP API Business Hub team",
    "email": "SAPAPIHubInfo@sap.com",
    "url": "https://api.sap.com/#/community"
  }
}
```

For guidelines on how to write helpful descriptions for the metadata of an API, see.

**Sample Code**

**API specification in YAML**

```yaml
host: localhost
schemes:
  - https
basePath: /espm-cloud-web/espm.svc/secure
```
Watch a video tutorial on how to define the metadata (title, version, description, contact) and the root URL (host, schemes, and basePath) for an API here.

Add Attributes

You can define additional fields or attributes in the Open API Specification to enhance the usability of your API. Swagger JSON provides additional attributes that start with x-, such as x-servers. They can be used to describe additional functionality that is not covered by the standard OpenAPI Specification. One such attribute is x-sap-shortText. The x-sap-shortText attribute is used to display a short description of your APIs in the APIs listing page of SAP API Business Hub.

API specification in YAML

```yaml
x-sap-shortText: A short description of your API
```

API specification in JSON

```json
{
  "x-sap-shortText": "A short description of your API"
}
```

To view the list of available additional attributes, see Additional Attributes in OpenAPI Specification [page 436].

Define Operations

Define the available paths (endpoints) and API operations. Paths are endpoints (resources) that your API exposes, such as /products or /store/inventory. Operations are the HTTP methods used to manipulate these paths, such as put, get, or post. Each operation is defined in its own operation object, including the path (endpoint), the HTTP method name, such as get or put, summary, and the description. You can add as many paths as you require.

API specification in YAML

```yaml
paths:
  /products:
    get:
      summary: Retrieves all products
      description: Retrieves the list of all available products in the inventory.
```

API specification in JSON

```json
{
  "paths": {
    "/products": {
      "get": {
        "summary": "Retrieves all products",
```
All API paths or endpoints are relative to the root URL, for example, /products means <scheme>://<host>/<basePath>/products. That is, if you want to list all the products available in the inventory, your API call would be:

GET http://localhost//espm-cloud-web/espm.svc/secure/products

For guidelines on writing helpful descriptions for API operations, see.

Define Parameters

Define the input parameters that you want your API Operation to accept. The information about the input parameters is provided in the Parameters object of each operation in the API definition file. Each parameter accepted by an operation is defined by a number of properties of the parameter object. The in property defines the location in which the parameter is passed.

---

### Sample Code

**API specification in YAML**

```yaml
parameters:
  description: |
    The workflow instance ID for which task instances are returned.
  in: query
  name: workflowInstanceID
  required: true
  type: string
```

### Sample Code

**API specification in JSON**

```json
{
  "parameters": {
    "description": "The workflow instance ID for which task instances are returned.",
    "in": "query",
    "name": "workflowInstanceID",
    "required": true,
    "type": "string"
  }
}
```

For a detailed explanation about how to define the input parameters for your API, see Adding Input Parameters - Headers and Queries [page 416]

For guidelines on writing helpful descriptions for input parameters, see.
Define Responses

Define the responses for API operations. For each possible response returned by an API operation, define a corresponding HTTP response status code and its description in the responses object. Defining HTTP status codes is crucial, as it describes the purpose of each method and the corresponding responses.

Sample Code

API specification in YAML

```yaml
responses:
  '204':
    description: |
      The task was successfully completed
      and the context was updated.
  '400':
    description: |
      Wrong format or structure
      of the provided request body.
  '403':
    description: |
      Access denied.
      You did not have the required permissions
      to access the resource.
  '404':
    description: |
      Not found. Possible reasons:
      - You provided a wrong URL.
      - The given task you are referring to doesn't exist.
      - You are not allowed to access the task.
  '422':
    description: |
      The workflow context provided in the
      request body contained invalid keys or values.
  '500':
    description: |
      Internal server error.
      The operation you requested
      led to an error during execution.
```

Sample Code

API specification in JSON

```json
{
  "responses": {
    "204": {
      "description": "The task was successfully completed and the context was updated."
    },
    "400": {
      "description": "Wrong format or structure of the provided request body."
    }
  }
}
```

For a detailed explanation about how to define responses for your API Operation, see Adding Responses [page 424].

Watch a video tutorial on how to define an API operation and its responses here.
For guidelines on writing helpful descriptions for API responses, see .

Add Definitions

If the same data type, parameter or response are used in multiple operations within the same API or service, you can simplify the API definition file by creating a reusable definition for each of them, and reference it where relevant across the API. for more information, see .

Add Security Definitions

Define all the implemented security mechanism for your APIs. Information about these schemes is provided under the Security Definitions object. The OpenAPI specification currently supports basic, apiKey, and oauth2 security scheme types.

Sample Code

```json
securityDefinitions:
  basicAuthentication:
    type: basic
  oauth2:
    type: oauth2
    description: > To use this API, you need to obtain the OAuth client credentials (client ID and secret) using the SAP HANA BTP cockpit.
    After that, pass these client credentials to the SAP HANA BTP token endpoint to obtain an access token.
    authorizationUrl: >-
      https://host1/oauth/tok/v1?grant_type=client_credentials
    flow: implicit
    scopes:
      product_view: Read Product entities
      product_manage: Manage Product entities
```

Sample Code

API specification in JSON

```json
{
  "securityDefinitions": {
    "basicAuthentication": {
      "type": "basic"
    }
  }
}
```

For more information, see Adding Security Definitions [page 432].

Add tags

To group related operations by categories, you can define tags to the operations so that they are rendered in an organized manner in the output. You define the tags under tags in the root swagger object. Each tag is identified by its name, which must be unique in the list of tags, and optional description and externalDocs. Then, you can assign the defined tags to operations, referring to them by names. Note that you can define a tag directly in an operation even if it is not defined on the root level.
Sample Code

API specification in YAML

tags:
- name: Task Instances
description: ''
- name: Workflow Definitions
description: ''
- name: Workflow Instances
description: ''
...

get:
tag:
- Task Instances
description: Retrieves the context of a task.

Sample Code

API specification in JSON

```json
{
  "tags": [
    {
      "name": "Task Instances",
      "description": ""
    },
    {
      "name": "Workflow Definitions",
      "description": ""
    },
    {
      "name": "Workflow Instances",
      "description": ""
    }
  ]
}
```

For more information on writing helpful description for tags, see .

Add External Links

Add external links, if any, in your API Definition to provide enhanced user assistance for using the APIs. You define the external links in the externalDocs object, which can be used on the root level for the entire API, and/or on the operation or tag level, as required.

Sample Code

API specification in YAML

```yaml
externalDocs:
  description: SAP Workflow Documentation
```
For more information, see [Creating APIs](#) [page 406] and [Perform Additional Tasks in API Designer](#) [page 446].

### 1.5.1.10.5.1.1 Adding Input Parameters - Headers and Queries

Define all input parameters for your API operation, irrespective of whether they are mandatory or optional. The parameters information can be modeled under the `parameters` definitions object in the OpenAPI specification.

In a RESTful API, the input parameters can be any of the following types:

- **Query Parameters**
  - When submitting a request, these parameters form the query string part at the end of the request URL. A question mark (?) character delimits the URL from the query string. For example, `Products?$top=2`. Multiple parameters can be supplied as name/value pairs in the format `name=value`. Each name/value pair is separated by the ampersand (&) character. For example, `Products?$skip=10&$top=2&someName=someValue`.

**Sample Code**

API specification in YAML

```yaml
parameters:
  - in: query
    name: pageNumber
```

Note

If you have created a new API, enter a name for it. If you are editing an existing API, save it under the existing name. When you save, you may receive warning messages about missing parameters. In this case, add the required parameters and save again.
type: integer
description: The page number from which the items must be listed.
- in: query
  name: pageNumber
  type: integer
  description: The page number from which the items must be listed.
- in: query
  name: limit
  type: pageSize
  description: The number of items to be returned on the specified page.

In the above example a GET call, say GET /products?pageNumber=2&pageSize=20 returns 20 products from page number 2.

**Sample Code**

API specification in JSON

```json
{
  "parameters": [
    {
      "in": "query",
      "name": "pageNumber",
      "type": "integer",
      "description": "The page number from which the items must be listed."
    },
    {
      "in": "query",
      "name": "limit",
      "type": "pageSize",
      "description": "The number of items to be returned on the specified page."
    }
  ]
}
```

In the above example a GET call, say GET /products?pageNumber=2&pageSize=20 returns 20 products from page number 2.

**Header Parameters**

Header parameters are components of the header section of an HTTP request or response. The name of a header field must be immediately followed by a colon : character. Any whitespace between the field name and the colon is syntactically incorrect. The header parameter value is provided as a plain text string. For example, Accept: application/json.

For example, suppose a call to GET /check requires the Request-ID header:

**Sample Code**

```
GET /check HTTP/1.1
Host: localhost
Request-ID: 2345-23567-4356-ab32-43ed
```

In the API designer, you will write the operation definition as follows:

**Sample Code**

API specification in YAML

```
/check:
  get:
    summary: Checks if the server is up.
```
parameters:
  - in: header
    name: Request-ID
    type: string
    required: true
responses:
  200:
    description: OK

---

## Sample Code

### API specification in JSON

```json
{
  "/check": {
    "get": {
      "summary": "Checks if the server is up.",
      "parameters": [
        {  
          "in": "header",
          "name": "Request-ID",
          "type": "string",
          "required": true
        }
      ],
      "responses": {
        "200": {
          "description": "OK"
        }  
      }
    }
  }
}
```

## Path Parameters

Path parameters are a flexible way of parameterizing the actual values used when creating the path to a resource. For example, if the value of the Product ID needs to be present in the path name, then this can be parameterized as follows: `/Products/{ProductId}`.

---

### Note

The parameter name must be the same as specified in the path. Also, remember to add `required: true`, because path parameters are always required.

---

## Sample Code

### API specification in YAML

```yaml
paths:
  /products{productId}:
    get:
      parameters:
        - in: path
          name: ProductId  # Note the name is the same as in the path
          required: true
          type: integer
          minimum: 1
          description: The product ID.
      responses:
        200:
          description: OK
```
You can familiarize more about defining parameters with the below examples:

The following APIs use OData protocol, and they support OData system queries. The commonly used system query parameters are defined in the following example:
Include count of elements.
ref [link](http://www.odata.org/documentation/odata-version-2-0/uri-conventions/#SystemQueryOptions) for more information
type: boolean

### Sample Code

**API Specification in JSON**

```json
{
  "parameters": {
    "top": {
      "name": "$top",
      "in": "query",
      "description": "Show only the first N elements, where N is a positive integer.",
      "type": "integer",
      "minimum": 0
    },
    "skip": {
      "name": "$skip",
      "in": "query",
      "description": "Skip the first N elements, where N is a positive integer as specified by this query option.",
      "type": "integer",
      "minimum": 0
    },
    "count": {
      "name": "$count",
      "in": "query",
      "description": "Include count of elements.",
      "type": "boolean"
    }
  }
}
```

These parameters can be referred in the `get` operation listed under `paths` as follows:

### Sample Code

**API Specification in YAML**

```yaml
paths:
  /Products:
    get:
      summary: Get entities from entity set Products
      description: Get entities from entity set Products
      security:
        - oauth2:
          - product_view
      tags:
        - Products
      parameters:
        - $ref: '#/parameters/top'
        - $ref: '#/parameters/skip'
        - $ref: '#/parameters/count'
```

### Sample Code

**API Specification in JSON**

```json
{
```
The advantage of defining the parameters in this way is that they can be defined once, and then later referenced from multiple places within the same specification file.

In the following example, other system queries such as $select, $expand and $orderby are defined at the operations level:

```
paths:
  /Products:
    get:
      summary: Get entities from entity set Products
      description: Get entities from entity set Products
      security:
        - oauth2:
            - product_view
      tags:
        - Products
      parameters:
        - $ref: '#/parameters/top'
        - $ref: '#/parameters/skip'
        - $ref: '#/parameters/count'
        - name: $orderby
          in: query
          description: Order by property values, for example `$orderby=Name` for sorting the Products by Name and `$orderby=Name desc` for sorting the Products by Name in descending order
          type: array
          uniqueItems: true
          items:
            type: string
```

Sample Code

API Specification in YAML
enum:
  - Category
  - Category desc
  - CategoryName
  - name: $select
  in: query
  description: >-
    Select properties to be returned. The value of a $select System Query Option is a comma-separated list of selection clauses. Each selection clause may be a Property name, Navigation Property name. for example `?$select=Category,Name` so that only Category and Name is returned
  type: array
  uniqueItems: true
  items:
    type: string
    enum:
      - Category
      - CategoryName
      - CurrencyCode
  - name: $expand
  in: query
  description: >-
    Expand related entities, The syntax of a $expand query option is a comma-separated list of Navigation Properties. for example `?$expand=Supplier` to get the related Supplier information inline.
    type: array
    uniqueItems: true
    items:
      type: string
      enum:
        - CustomerReview
        - Supplier

---

Sample Code

API Specification in JSON

```json
{
  "paths": {
    "/Products": {
      "get": {
        "summary": "Get entities from entity set Products",
        "description": "Get entities from entity set Products",
        "security": [
          
        ],
        "tags": [
          "Products"
        ],
        "parameters": [
          {"$ref": "#/parameters/top"},
          {"$ref": "#/parameters/skip"},
          {"$ref": "#/parameters/count"}
        ],
```
In the following example, a paths parameter called `ProductId` is defined as a required field:

```
paths:
  /Products/{ProductId}:
    get:
      summary: Get entity from Products by key.
      description: Returns the entity with the key from Products
      tags:
        - Products
      parameters:
        - name: ProductId
          in: path
          required: true
```
**Sample Code**

API Specification in JSON

```json
{
    "paths": {
        "/Products{ProductId}": {
            "get": {
                "summary": "Get entity from Products by key.",
                "description": "Returns the entity with the key from Products",
                "tags": [
                    "Products"
                ],
                "parameters": [
                    {
                        "name": "ProductId",
                        "in": "path",
                        "required": true,
                        "description": "key: ProductId",
                        "type": "string"
                    }
                ]
            }
        }
    }
}
```

---

**Related Information**

**Parameters Definitions Object**

### 1.5.10.5.1.2 Adding Responses

Define all the expected responses, including the response headers, response message, and error messages. The server receives an incoming request and responds with a message indicating whether the request was successful or not. Such a response consists of the following elements:

- An HTTP status code
- Zero or more Response Headers
- An optional Response Body

Use the standard HTTP status codes for describing the success or failure of the request. It is also recommended to use the HTTP response headers.
1.5.1.10.5.1.2.1 Adding Responses without a Body

It is a valid and common feature of a RESTFul API that the response to certain operations contains no body. For instance, the Update operations (implemented by the HTTP method PUT) belonging to the ESPM OData API in the example below typically return an HTTP status code 204 with no message body.

In the example below, the same information is modeled using the Open API specification:

```
---
Sample Code
API specification in YAML

paths:
  '/Products('{ProductId}')':
    put:
      summary: Update a product entity in Products entityset
      description: Update a product entity in Products entityset
      tags:
        - Products
      parameters:
        - name: ProductId
          in: path
          required: true
          description: 'key: ProductId'
          type: string
        - name: Product
          in: body
          description: The entity to patch
          schema:
            $ref: '#/definitions/Product'
      responses:
        '204':
          description: Empty response
---

---
Sample Code
API specification in JSON

{
  "paths": {
    "/Products('{ProductId}')": {
      "put": {
        "summary": "Update a product entity in Products entityset",
        "description": "Update a product entity in Products entityset",
        "tags": [
          "Products"
        ],
        "parameters": [
          
```
1.5.10.5.1.2.2 Responses with a Body

For certain operations like Create (implemented by the HTTP method POST), the newly created resource is typically returned as the response body. This is to save the client from then having to perform a subsequent Read operation to determine the exact server-side state of the newly created resource.

In the example below, this information is modeled using the Open API specification. Since the definition of the Product returned in the response is exactly the same as that previously defined in the section on requests, the reference to the Product definition can be reused.

**Sample Code**

API specification in YAML

```yaml
paths:
  '/Products('{ProductId}')':
    post:
      summary: Create a product
      description: Create a product entity in Products entityset
      tags:
        - Products
      parameters:
        - name: Product
          in: body
          description: Product entity to be created
          schema:
            $ref: '#/definitions/Product'
      responses:
        '201':
          description: Created product
          schema:
            $ref: '#/definitions/Product'
```

1.5.1.10.5.1.2.3 Response Headers

Along with the response message, the server can respond with zero or more headers.

In the example below, the custom HTTP header DataServiceVersion is described. This header informs the client about the version of OData used to build the response.

```
paths:
  '/Products('{ProductId}')':
    post:
      summary: Create a new product
      description: Post a new entity to entity set Products
      tags:
        - Products
      parameters:
        - name: Product
          in: body
          description: Product entity to be created
          schema:
            $ref: '#/definitions/Product'
      responses:
        201:
          description: Created product
          schema:
            $ref: '#/definitions/Product'
```

Based on such a definition, the client can then expect to receive a response containing at least the `DataServiceVersion` header. For example, a Gateway OData server returns the following headers. Notice that `DataServiceVersion` is among those headers.
i Note

Notice also that all the HTTP header fields returned from an ABAP OData server are in lowercase. This conforms to the HTTP standard that all HTTP header fields are case insensitive.

1.5.10.5.1.2.4 Error Response

In case of an error, the server should return an error message that best describes the cause of the problem and, where possible, provides information on how to correct that problem. It is a good practice to use standard HTTP error status codes to inform about the nature of the error and have a common error schema describing each of these status codes.

The text of error message should always help move the user towards the solution. Simply responding with a generic error message such as “Internal server error” leaves the user stranded and unable to proceed towards a solution.

For example, the ESPM OData API returns errors using the OData error format, and therefore we have documented some of the commonly returned error codes. Since the error messages are the same for all the operations, we have documented them in one place using the responses object of the Open API specification, then referenced those definitions within the same file.

Sample Code

API specification in YAML

```yaml
responses:
  401:
    description: Unauthorized
  404:
    description: Not Found
    schema:
      $ref: '#/definitions/odata.error'
  400:
    description: Bad Request
    schema:
      $ref: '#/definitions/odata.error'
  500:
    description: Internal server error
    schema:
      $ref: '#/definitions/odata.error'

definitions:
  odata.error:
    type: object
```
properties:
  code:
    type: string
    description: Error code
  message:
    type: object
    properties:
      lang:
        type: string
        description: Language code of the error message
      value:
        type: string
        description: Detailed error message

---

Sample Code
API specification in JSON

```json
{
  "responses": {
    "400": {
      "description": "Bad Request",
      "schema": {
        "$ref": "#/definitions/odata.error"
      }
    },
    "401": {
      "description": "Unauthorized"
    },
    "404": {
      "description": "Not Found",
      "schema": {
        "$ref": "#/definitions/odata.error"
      }
    },
    "500": {
      "description": "Internal server error",
      "schema": {
        "$ref": "#/definitions/odata.error"
      }
    }
  },
  "definitions": {
    "odata.error": {
      "type": "object",
      "properties": {
        "code": {
          "type": "string",
          "description": "Error code"
        },
        "message": {
          "type": "object",
          "properties": {
            "lang": {
              "type": "string",
              "description": "Language code of the error message"
            },
            "value": {
              "type": "string",
              "description": "Detailed error message"
            }
          }
        }
      }
    }
  }
}
```
In the example below, we show the usage of the above error responses within an operation to update a Product entity.

## Sample Code

### API specification in YAML

```yaml
paths:
  '/Products('{ProductId}')':
    put:
      summary: Update entity in EntitySet Products
      description: Update entity in EntitySet Products
      tags:
        - Products
      parameters:
        - name: ProductId
          in: path
          required: true
          description: 'key: ProductId'
          type: string
        - name: Product
          in: body
          description: The entity to patch
          schema:
            $ref: '#/definitions/Product'
      responses:
        '204':
          description: Empty response
        401:
          $ref: '#/responses/401'
        404:
          $ref: '#/responses/404'
        400:
          $ref: '#/responses/400'
        500:
          $ref: '#/responses/500'
```

### Sample Code

### API specification in JSON

```json
{
  "paths": {
    "/Products('{ProductId}')": {
      "put": {
        "summary": "Update entity in EntitySet Products",
        "description": "Update entity in EntitySet Products",
        "tags": [
          "Products"
        ],
        "parameters": [
          {
            "name": "ProductId",
            "in": "path",
            "required": true,
            "description": "key: ProductId",
            "type": "string"
          },
          {
            "name": "Product",
            "in": "body",
            "description": "The entity to patch",
            ...
          }
        ],
        ...
      }
    }
  }
}
1.5.1.10.5.1.3 Adding Security Definitions

Security definitions allow you to define various authentication types (security schemes) supported by an API. If you want to implement any authentication mechanism for your API, then we recommended you to define and apply the associated authentication types in your API.

OpenAPI specification lets you define the following authentication types for an API:

- **basic**
  Used in situations where simple userid/password based authentication is sufficient

- **apiKey**
  Used in situations where the application must authenticate itself by presenting an API Key value as part of every request. This type of authentication is independent of any user authentication.

- **oauth2**
  Used when the Oauth2 based authentication is required.

Authentication types are described using the `securityDefinitions` and `security` properties of the OpenAPI specification. Use the `securityDefinitions` property to define all authentication types supported by the API, and then use the `security` property to apply specific authentication types to the whole API.

In the API below, it supports three security schemes named `basicAuthentication`, `apiKeyAuth`, and `OAuth2`. These names are used to refer to these security schemes from elsewhere within the API.

---

**Sample Code**

API specification in YAML

```yaml
info:
  title: >-
```

This sample is a reference service, showcases the e-commerce APIs for products, and suppliers.

securityDefinitions:
  basicAuthentication:
    type: basic
  apikeyAuth:
    type: apiKey
    in: header
    name: X-API-Key
  OAuth2:
    type: oauth2
    description: >-
      To use this REST API, you need to get OAuth client credentials (client ID and secret) using the SAP BTP Cockpit. After that, you need to pass the obtained client credentials to the SAP BTP token endpoint to obtain an access token.
      tokenUrl: https://api.hana.ondemand.com/oauth2/apitoken/v1?
      grant_type=client_credentials
      flow: application
      scopes:
        product_view: Read Product entities
        product_manage: Manage Product entities

After you have defined the security schemes in the securityDefinitions property, you can apply them to the whole API using the security property on the root level. When used on the root level, security applies the defined security definitions globally to all the operations within the API. In the following example, the API calls can be authenticated using either basic authentication (username and password) or API key.

Sample Code

API specification in JSON

```json
{
  "info": {
    "title": "This sample is a reference service, showcases the e-commerce APIs for products, and suppliers."
  },
  "securityDefinitions": {
    "basicAuthentication": {
      "type": "basic"
    },
    "apikeyAuth": {
      "type": "apiKey",
      "in": "header",
      "name": "X-API-Key"
    },
    "OAuth2": {
      "type": "oauth2",
      "description": "To use this REST API, you need to get OAuth client credentials (client ID and secret) using the SAP BTP Cockpit. After that, you need to pass the obtained client credentials to the SAP BTP token endpoint to obtain an access token."
    },
    "tokenUrl": "https://api.hana.ondemand.com/oauth2/apitoken/v1?grant_type=client_credentials",
    "flow": "application",
    "scopes": {
      "product_view": "Read Product entities",
      "product_manage": "Manage Product entities"
    }
  }
}
```
This sample is a reference service that showcases the e-commerce APIs for products, and suppliers.

securityDefinitions:
  basicAuthentication:
    type: basic
  apikeyAuth:
    type: apiKey
    in: header
    name: X-API-Key
  OAuth2:
    type: oauth2
    description: "To use this REST API, you need to get OAuth client credentials (client ID and secret) using the SAP BTP cockpit. After that, you need to pass the obtained client credentials to the SAP BTP token endpoint to obtain an access token."
    tokenUrl: https://api.hana.ondemand.com/oauth2/apitoken/v1?grant_type=client_credentials
    flow: application
    scopes:
      product_view: Read Product entities
      product_manage: Manage Product entities
  security:
    - basicAuthentication: []
    - apikeyAuth: []

This sample is a reference service, which showcases the e-commerce APIs for products, and suppliers.

"securityDefinitions": {
  "basicAuthentication": {
    "type": "basic"
  },
  "apikeyAuth": {
    "type": "apiKey",
    "in": "header",
    "name": "X-API-Key"
  },
  "OAuth2": {
    "type": "oauth2",
    "description": "To use this REST API, you need to get OAuth client credentials (client ID and secret) using the SAP BTP cockpit. After that, you need to pass the obtained client credentials to the SAP BTP token endpoint to obtain an access token."
    "tokenUrl": "https://api.hana.ondemand.com/oauth2/apitoken/v1?grant_type=client_credentials",
    "flow": "application",
    "scopes": {
      "product_view": "Read Product entities",
      "product_manage": "Manage Product entities"
    }
}
The security scheme applied at the root level can be overridden in individual operations. In the following example, `basic` is defined and applied as security scheme at the root level. Whereas, for `/products`, the security scheme defined at the root level is overridden by having no security scheme.

### Sample Code

**API specification in YAML**

```yaml
securityDefinitions:
  basicAuthentication:
    type: basic
# To apply Basic auth to the whole API:
security:
  - basicAuthentication: []
paths:
  /something:
    get:
      summary: Get entities from entity set Products
      description: Get entities from entity set Products
      # To apply Basic auth to an individual operation:
      security: [] #No security
      responses:
        200:
          description: OK (successfully authenticated)
```

**API specification in JSON**

```json
{
  "securityDefinitions": {
    "basicAuthentication": {
      "type": "basic"
    }
  },
  "security": [],
  "paths": {
    "/something": {
      "get": {
        "summary": "Get entities from entity set Products",
        "description": "Get entities from entity set Products",
        "security": [],
        "responses": {
          "200": {
            "description": "OK (successfully authenticated)"
          }
        }
      }
    }
  }
}
1.5.10.5.1.4 Additional Attributes in OpenAPI Specification

OpenAPI Specification allows you to define additional attributes or custom extensions that start with `x-`, such as `x-servers`. They can be used to describe extra functionality of the API that is not covered by the standard OpenAPI Specification.

Additional attributes or custom extensions are supported on the root level of the OpenAPI Specification and in the following places:

- `info` section
- `paths` section
- `responses` section
- `tags`
- Security schemes

The following additional attributes have been introduced for use with API designer. All these attributes must be defined at the root level of the OpenAPI Specification.

**List of additional attributes**

**x-sap-api-type**

You can use this attribute to display the API type in the APIs listing page of SAP API Business Hub. For example:

```
{
  "x-sap-api-type": "ODATA"
}
```

The API Type is set according to the following rules of precedence:

1. If EDMX file is present, then the type is set as ODATA.
2. If WSDL file is present, then the type is set as SOAP.
3. If `x-sap-api-type` is mentioned in the swagger, then the respective value is set. For OData APIs, to differentiate the V2, and V4 OData APIs, you will need to specify the following values in swagger for the `x-sap-api-type` attribute:

<table>
<thead>
<tr>
<th>Type of API</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>OData V2 API</td>
<td>ODATA</td>
</tr>
<tr>
<td>OData V4 API</td>
<td>ODATAV4</td>
</tr>
</tbody>
</table>

4. The default is REST.
x-sap-shortText

The x-sap-shortText attribute is used to display a short description of your APIs in the APIs listing page of SAP API Business Hub. This short text description appears below the title in the list of APIs/services on the Artifacts page in SAP API Business Hub.

**Note**

The x-sap-shortText is a mandatory attribute that must be defined in the API definition file to enable publication of your API on the SAP API Business Hub.

**Sample Code**

```json
{
   "info": {
      "title": "SAP Workflow service API",
      "description": "Provides functionality to work with SAP Workflow service. You can, for example, start new workflow instances and work with tasks."
   },
   "x-sap-shortText": "Work with the SAP Workflow service."
}
```

x-sap-stateInfo

You use x-sap-stateInfo attribute to display the current status of an API that you want to publish on SAP API Business Hub. An API can be marked with any of the following statuses:

- Beta
- Active
- Deprecated
- Decommissioned

**Note**

The x-sap-stateInfo is an optional attribute. That is, if you do not use this attribute in your API definition file, then by default, the current status of an API is marked as Active. However, if you want to publish your API in the Beta status or you have decided to transition your API from Active to Deprecated or Decommissioned status, then it is mandatory to use this attribute to indicate the new status of your API.

The following is a sample code snippet of an API definition file in which x-sap-stateInfo attribute is used to indicate that the current status of the API is Beta.

**Sample Code**

```json
{
   "swagger": "2.0",
   "info": {
      "title": "Business API",
      "version": "1.1.0",
      "description": "API for Reading Business Partner, Supplier, Customer and Contact Persons"
   },
   "x-sap-stateInfo": {
      "state": "Beta"
   }
}
```
The following is a sample code snippet of an API definition file in which `x-sap-stateInfo` attribute is used to indicate that the current status of the API is deprecated as on 14th August 2018, and a new version of the API is available:

```json
{
    "swagger": "2.0",
    "info": {
        "title": "Business API",
        "version": "1.1.2",
        "description": "API for Reading Business Partner, Supplier, Customer and Contact Persons"
    },
    "x-sap-stateInfo": {
        "state": "Deprecated",
        "deprecationDate": "14 Aug 2018",
        "successorApi": "http://api.sap.com/api/product_text_classification_api"
    }
}
```

**Note**

You must enter the `deprecationDate` in the format `dd mmm yyyy`, where `mmm` is a string indicating the month. For example, Jan, Jul, Nov etc.

If `x-sap-stateInfo` attribute is not defined or left empty, then the default status of the API is taken as `Active`.

If you have defined the `x-sap-stateInfo` attribute, then you must also ensure that you have entered the change log information in the `Artifact.json` file of your API package.

**Note**

If you do not enter the changelog information in `Artifact.json` file, then it results in unsuccessful builds.

The following is a sample code snippet of the `Artifact.json` file in which `changelog` attribute is used to indicate the most recent state of the API.

```json
{
    "type": "API",
    "changelog": [
        {
            "state": "Deprecated",
            "date": "14 Aug 2018",
            "version": "1.0.0",
            "notes": "New api with enhanced functionality is available"
        },
        {
            "state": "Active",
            "date": "18 Jan 2018",
            "version": "1.0.0",
            "notes": "Some bug fixes and performance enhancement"
        }
    ]
}
```
You must enter the date in the format dd mmm yyyy, where mmm is a string indicating the month. For example, Jan, Jul, Nov etc. The most recent state of the API must appear as the first entry under changelog, and the values defined for the state attribute in API definition file and Artifact.json file must be same. It is a good practice to indicate the recent changes made to the API using the notes attribute. This will help your API consumers to know if they need to follow certain rules or conditions before using the API.
The following is a sample code snippet of an API definition file in which `x-sap-stateInfo` attribute is used to indicate that the current status of the API is decommissioned as on 05th September 2018, and a new version of the API is available:

```json
{  
  "swagger": "2.0",
  "info": {
    "title": "Business API",
    "version": "1.1.2",
    "description": "API for Reading Business Partner, Supplier, Customer and Contact Persons"
  },
  "x-sap-stateInfo": {
    "state": "Decommissioned",
    "decommissionedDate": "05 Sep 2018",
    "successorApi": "http://api.sap.com/api/product_text_classification_api"
  }
}
```
The following is a sample code snippet of the Artifact.json file in which changelog attribute is used to indicate the decommissioned state of the API.

### Note

When you transition an API from one state to another, you must enter the changelog information in Artifact.json file. SAP Content pipeline strongly recommends the following:

- You add the changelog information if you are publishing the first or initial version of an API.
- You add the changelog information for every consecutive update of an already published API.

#### Sample Code

```json
{
  "type": "API",
  "changelog": [
    {
      "state": "Decommissioned",
      "date": "05 Sep 2018",
      "version": "1.1.2",
      "notes": "This API is decommissioned"
    },
    {
      "state": "Deprecated",
      "date": "18 Jul 2018",
      "version": "1.0",
      "notes": "This API is deprecated"
    },
    {
      "state": "Active",
      "date": "18 Jan 2018",
      "version": "1.0.0",
      "notes": "Some bug fixes and performance enhancement"
    }
  ]
}
```

### Note

You must enter the date in the format dd mmm yyyy, where mmm is a string indicating the month. For example, Jan, Jul, Nov etc. The most recent recent state of the API must appear as the first entry under changelog, and the values defined for the state attribute in API definition file and Artifact.json file must be same. It is a good practice to indicate the recent changes made to the API using the notes attribute. This will help your API consumers to know if they need to follow certain rules or conditions before using the API.
The following images shows a sample API, which is marked Decommissioned on the SAP API Business Hub:

You can also choose to mark an API operation or a parameter as deprecated.

**Note**

In OpenAPI 2.0 specification, you have the provision to mark only an API operation as deprecated whereas in OpenAPI 3.0 specification, you can mark both API operation and a parameter as deprecated.

In OpenAPI 2.0, use `deprecated: true` to mark an API operation as deprecated.

```
operation deprecation

paths:
  /list:
    get:
      responses:
        '200':
          description: This API Operation is deprecated.
```
In OpenAPI 3.0, use `deprecated: true` to mark an API operation and parameter as deprecated.

**Sample Code**

**operation deprecation**

```yaml
paths:
  /list:
    get:
      responses:
        '200':
          description: This API Operation is deprecated.
          deprecated: true
```

**Sample Code**

**parameter deprecation**

```
- in: query
  name: id
  required: true
  schema:
    type: string
  deprecated: true
  description: Deprecated, use 'itemId' parameter instead.
```

For more information, see [API Deprecation Policy](https://example.com/api-deprecation-policy).

**x-sap-csrf-token-path**

You use this attribute to provide a path relative to the basepath of your API for fetching X-CSRF-Token. That is, this attribute must contain the path of a resource that handles the fetching of x-csrf token requests. The relative path provided must not include server and transfer protocol information.

**Sample Code**

```
{
  "host": "api.workflow.com",
  "schemes": [
    "https"
  ],
  "basePath": "/workflow-service/rest",
  "x-sap-csrf-token-path": "/<relative_path_for_handling_csrf_token_fetch>"
}
```

**Note**

The `relative_path` must be a path relative to the basepath that you have provided in your API.

**Sample Code**

**Example**

```
{
  "host": "api.workflow.com",
```
"schemes": ["https"],
"basePath": "/workflow-service/rest",
"x-sap-csrf-token-path": "/v1/xsrf-token"
}

**x-sap-software-min-version**

You use this attribute to provide the minimum software version. For example:

```json
{  "x-sap-software-min-version": "SAP NetWeaver 2.0"
}
```

**x-sap-ext-overview**

You can use this attribute to provide stakeholder-specific information. For example:

```json
{
  "x-sap-ext-overview": [{
    "name": "Communication Scenarios",
    "values": ["SAP_COM_0025 Name of SAP_COM_0025", "SAP_COM_0028 Name of SAP_COM_00028"]
  },
  {
    "name": "Additional Property",
    "values": ["EntryValue1", "EntryValue2", "EntryValue3"]
  }]
}
```

→ Remember

Authentication details must not be a part of the x-sap-ext-overview. Add authentication details in the security scheme section. For more information, see .

**x-servers**

The Open API specification 2.0 does not support multiple hosts (and ports), neither are path templating or patterns supported. Some APIs need support for both multiple hosts and path templating in the host parameter. This is because the host and landscape vary between regions.

These features will be supported in the servers property in the Open API specification v3.0. However, in Open API specification v2.0, the required configuration values can be added via the custom extension x-servers.

For more information about how to specify sandbox url and multiple hosts or production servers in OpenAPI 3.0, see the Sandbox and Configure Information sections in

See here to know more information about the differences between OpenAPI 2.0 and Open API 3.0 specifications

If values are supplied for host, schemes, and basepath, then they together form the root URL of the API. The root URL points to a Sandbox system or a test system wherein the API can be tested.
"Sample Code"

```json
{
  "host": "sandbox.api.sap.com",
  "schemes": [
    "https"
  ],
  "basePath": "/sap/v1"
}
```

For more information on how to define schemes, host and basepath in the API specification, see Creating APIs [page 406].

**Note**

The Try it Out! feature on SAP API Business Hub can be configured to enable API testing experience either in a sandbox system or in a productive system. To try out the APIs in a sandbox system, the host attribute in the OpenAPI specification must be defined and point to the API Sandbox URL. Similarly, to try out the APIs in a productive system, the x-servers attribute in the OpenAPI specification must be defined and point to the URL of the productive system. For example, the productive system can be URL to an application/service running on an active SAP BTP account.

The x-servers attribute is used when you want to specify multiple hosts, for example, to specify values for different servers located across various geographical boundaries. The example below shows how multiple hosts with path templates can be defined using x-servers attribute.

```json
{
  "info": {
    "title": "This sample is a reference service, which runs on SAP BTP showcasing the e-commerce APIs for products, and suppliers.",
    "version": "1.0"
  },
  "x-servers": [
    {
      "url": "https://{appname}{accountname}.{landscapehost}/espm-cloud-web/espm.svc/secure",
      "description": "ESPM OData endpoints",
      "templates": {
        "appname": {
          "default": "espm",
          "description": "The application name used while deploying the ESPM application"
        },
        "accountname": {
          "description": "The SAP BTP subaccount id where the application is deployed"
        },
        "landscapehost": {
          "enum": [
            "hana.ondemand.com",
            "us1.hana.ondemand.com",
            "us2.hana.ondemand.com",
            "ap1.hana.ondemand.com",
            "hanatrial.ondemand.com"
          ],
          "default": "hana.ondemand.com",
          "description": "The region(host) where the application is deployed."
        }
      }
    }
  ]
}
```
For more information on how x-servers attribute is used for enabling API test experience in a productive environment in SAP API Business Hub, see here.

1.5.1.10.5.1.5 Perform Additional Tasks in API Designer

Besides creating new APIs or editing existing APIs in the API designer, you can also do the following:

- To model Open API JSON content in the designer, choose Paste JSON.
- To model an Open ODATA API, choose Paste OData Metadata.
- To convert an API written in RAML to Open API, choose Paste RAML.
- To import a YAML or JSON format file from your local server, choose Import.
- To download the API swagger specifications, choose Download and select JSON or YAML format.
- You can generate a server stub from the API definition file. The generated server stub can be used for deploying applications locally and as well as on Cloud Foundry.
  - From the Generate Server Stub dropdown menu, choose the required language in which you want to generate the server stub.
  - In the Project Metadata dialog window, enter the following information:
    - Package Name: The name of the package.
    - GroupId: The ID of the project’s group.
    - Artifact: The ID of the artifact (project).
    - Artifact Version: The version of the artifact under the specified group.
  - Choose Generate Project.
  - The server stub is downloaded in a ZIP file. The generated server code contains stub methods and a README.md file with all the information required for building applications. If you have generated a server stub for Cloud Foundry deployment, then the README.md file contains instructions for deploying application on Cloud Foundry. Each language creates a different README file. Go through it to learn about how to build and deploy the application.
  - Choose Save to save the modeled API. You can choose to save the modeled API with a version by choosing the option Save as Version.

1.5.10.6 Copy an API

You can copy an API proxy in the same subscription.

Prerequisites

You are assigned with the required roles.
Context

To copy an API proxy, proceed as follows:

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. On the APIs tab page, choose the Action icon against the required API and then select the Copy option. Alternatively, you can open the required API and in the details page select the option Copy.
4. In the Copy API dialog box, the details for each attribute is pre filled. Name and API Base Path fields should be unique. So, it is required to change the API name and basepath values. The values for the remaining fields can either be retained or changed.
   a. Optional: Enter a version for your API proxy.
      When you choose to version your API proxy, its name will be appended with the version, and its basepath will be prepended with the version. For example, if the version you enter is v1, the name will be Name_v1, and the basepath will be /v1/SalesOrder. For more information, see API Versioning [page 391]
      For more details on each field, see Create an API [page 393].
5. Choose Copy.
6. After you have copied the API, you can select one of the following two actions for the API:

<table>
<thead>
<tr>
<th>Action</th>
<th>Resulting API State</th>
<th>Future Action on API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save as Draft</td>
<td>Not Deployed</td>
<td>Deploy</td>
</tr>
<tr>
<td></td>
<td>API is available only in the API portal, and is not available for product assignments.</td>
<td>API is deployed and is ready for product assignments.</td>
</tr>
<tr>
<td>Save and Deploy</td>
<td>Deployed</td>
<td>Undeploy</td>
</tr>
<tr>
<td></td>
<td>Only deployed APIs can be selected for product publishing.</td>
<td>If any API is undeployed after being published, it is removed from the developer portal. When the API is deployed again, the product is updated. You can bring down an API without having to delete it from the product assignment. You cannot undeploy an API if it is the only one associated with the product.</td>
</tr>
</tbody>
</table>

i Note

An API proxy consists of a virtual host and a base path. The base path can be identical for multiple API proxies, provided API proxies have different virtual hosts. This means, for an API proxy, the combination of the virtual host and base path should be unique.
The example below explains the same, where AP1 is proxy 1, AP2 is proxy 2, VH1 is Virtual Host 1, VH2 is the Virtual Host 2, and BP(A) is the base path.

Example: AP1 = VH1 + BP(A) AP2 = VH2 + BP(A)

Related Information

- Import an API [page 448]
- Export an API [page 450]
- Create an API from API Designer [page 405]

### 1.5.1.10.7 Import an API

This topic describes how to import an existing API definition into the API portal.

#### Prerequisites

You are assigned with **API Portal. Administrator** role.

The API proxy content is available as a .zip file and swagger json file. The contents adhere to the API proxy structure as defined in **API Proxy Structure [page 352]**.

---

**Note**

- Ensure that the API proxy name in the `<APIProxyName>.xml` file and the value of the `base_path` field (inside the APIProxyEndpoint file) is unique.
- **APIResources**, **DocumentationFileResource**, and **Policy** folders are not required in the .zip file.
- Ensure that the resource documentation is available in a single **OAS_json** file. Note that you cannot refer to external links in the API definitions within this json file.
- API Management supports import of API definitions in both OAS 2.0 and OAS 3.0. To know more about OAS 3.0 support in API Management, see **OpenAPI Specification 3.0 [page 82]**.
- Optional: If you want to mention the API State of the API proxy you are importing, you need to update the following in the API proxy XML:
  - API State: The state of the API proxy. To know more about API States, see **API Proxy States [page 393]**
  - Successor API: If the API state of the API proxy you are importing is deprecated or decommissioned, you need to provide information about a successor API that the consumer should use.
  - Release Metadata: If the API state of the API proxy you are importing is deprecated or decommissioned, use this field to provide information about the following:
    - Reason for deprecation or decommissioning
    - Date of deprecation or decommissioning
External successor API: If you provide an external successor API, you will not use the Successor API parameter.

Sample Code

```xml
<APIState>Deprecated</APIState>
<successorAPI>XYZ</successorAPI>
<releaseMetadata>"reason":"undefined","date":1603132200000,"externalSuccessorAPI":"","changed_at":1602578356857,"changed_by":"xxx@abc.com"</releaseMetadata>
```

**File Resource** is a script or code snippet that can be attached to Flows using policies. An API proxy container supports definition of a number of Java, Python or XSL scripts. These scripts can be executed in the context of either a Java Script, Python Script or XSL Transformation policy. Once a Script is defined, it can be applied as a either a Java Script policy, Python Script policy or XSL Transform policy in different Flows.

**Context**

API Management provides the option to import an API definition.

**Note**

When an API proxy is transported or exported individually or as a part of a Product, by default, it gets imported to the target in the deployed state.

**Procedure**

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. On the **APIs** tab page, choose ![](Create) Import API.
4. In the Import API window:
   a. Choose Browse. Navigate and choose the required API from your local file system.
      You can attach files of type .zip and .json. When you import the API, you can create a new API or replace the existing API across landscapes seamlessly.
      You can include the configurations for health monitor and load balancer in the .zip file. For more information, see Load Balancing Across API Providers [page 476].
   b. You can choose to import either an API whose lifecycle will be managed by SAP API Management, or an externally managed API, when importing the json file. Choose Manage this API to import an API whose lifecycle is to be handled by API Management.
   c. Select a virtual host alias from the Virtual Host drop down.
d. If you want to version your API when uploading the .json file, select Yes for the Create a Version toggle button (this is set to No by default), and enter the version in the Version field that appears. The version will be appended to the name and prepended to the base path of the API, to create a unique API proxy version. For example, If the version you enter is v1, the name will be Name_v1, and the basepath will be /v1/SalesOrder. To know more about versioning your API, see API Versioning [page 391].

e. If you want to list an externally managed API, when uploading the json file, choose List as Externally Managed API. This API will only be listed in API Management, and it’s lifecycle will not be managed. For more information, see Externally Managed APIs [page 82].

5. Choose OK.

Related Information
Export an API [page 450]

1.5.1.10.8 Export an API

Once you create an API in the API portal, you can choose to export it.

Prerequisites

You have created an API in the API portal with:

- You are assigned with APIPortal. Administrator role.
- Relevant resources and documentation.
- Policies attached to the API.

Context

To export an API proxy, proceed as follows:

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. On the APIs tab page, choose the Action icon against the required API and then select the Export option.
4. Alternatively, you can open the required API, choose the breadcrumb and then select the option Export.
5. A .zip file is exported with contents as described in the API Proxy Structure [page 352]. All the content related to the API documentation is available in Swagger.json file. The current state of the API, will also be available in the exported zip.

Related Information

Import an API [page 448]

1.5.1.10.9 Create a Policy

Define a policy to set rules on the API, for example, to enforce security or control API traffic.

Prerequisites

- You have a thorough understanding on Policies and the various Flows it can be attached to. For more information, see Policies [page 92].
- You are familiar with the different types of policies supported by SAP API Management. For more information see, Policy Types [page 93].
- You have the payload of the policy you want to create.

Procedure

**Context:** You are creating an API proxy and want to add a policy to it.

1. While creating an API proxy, choose Policies on the details screen.
2. Select a Flow on which you want to apply the policy.

<table>
<thead>
<tr>
<th>i Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flows are available in the top left section of the Policy Designer. You can select an existing flow or create a conditional Flow. To create a conditional Flow, choose the icon + beside the ProxyEndpoint or TargetEndpoint depending on which endpoint you want to assign the policy. Enter a name for the conditional Flow.</td>
</tr>
</tbody>
</table>

3. From the Policies section right hand side, choose the icon + (Add) beside the required policy.
4. In the Create Policy pop-up, enter a name for your policy.
5. From the Stream drop-down, select the processing pipeline where this policy should be assigned:
   - Incoming Request
   - Outgoing Response
6. Choose Add.
The specified policy is created and denoted in the Policy Designer. A sample payload for the selected policy is added in the editor.

7. In the Conditional String field, specify the condition on which this policy should be executed.
8. In the editor below, provide the payload for the selected policy.
9. Choose Save.

The policy now appears under the Created Policies section. The count beside the Flow to which this policy was assigned, is incremented.

If required, the policy name can be edited in the policy editor. However, if you edit the policy name, then you need to change the policy name at instances where the policy name is referred.

1.5.10.10 Create a Script

This topic describes how to create a FileResource (also called Script).

Prerequisites

- You are familiar with the concept of Scripts. For more information, see File Resource [page 352].
- You have the payload of the Script that you want to create.

Context

Script is a FileResource or code snippet that is attached to Flows using policies. SAP API Management supports the creation of JavaScript, Python, and XSL scripts.

Context: You are creating an API proxy and want to add a script to it.

Procedure

1. While creating an API proxy, navigate to the Policies on API tab page.
2. Select Invoke Policy Designer.
3. Select the icon + (Add) beside the Scripts section at the bottom-left of the Policy Designer.
4. Enter a name for the script.
5. From the Type field, select one of the following options:
   - JavaScript
   - Python
   - XSL
6. Choose Add.

The added script appears under the Scripts section.
7. Select the Script you created and provide the script details in the editor.
8. Choose Save.

You can reference the scripts from a Java Script policy, Python policy or XSL Transform policy.

**1.5.1.10.11 Create a Policy Template**

Create a policy template add it to an API proxy.

**Prerequisites**

- You have a thorough understanding of policies and the various flows they can be attached to. For more information, see Policies [page 92].
- You are familiar with the different types of policies supported by SAP API Management. For more information, see Policy Types [page 93].
- You have the payload of the policy you want to create.

**Procedure**

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
   A list of registered APIs appears in the catalog.
3. In the list, click the API for which you want to create the policy template.
5. On the Policy Editor screen, choose Policy Template > Create
6. In the Create Policy Template window, proceed as follows:
   - Enter a name for the template in the Name field.
   - Enter a description for the template in the Description field.
   - Select the required policies from the Policies Available list.
7. Choose OK.

To view the policy template that you have just created, navigate from the API Portal home page to Develop -> Policy Templates.

**Related Information**

*Apply a Policy Template* [page 454]
1.5.10.11.1 Apply a Policy Template

Apply a policy template to an API.

Prerequisites

- You have a thorough understanding of policies and the various flows they can be attached to. For more information, see Policies [page 92].
- You are familiar with the different types of policies supported by SAP API Management. For more information, see Policy Types [page 93].
- You have the payload of the policy you want to create.

Context

You are applying a policy template and want to apply it to an API proxy.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
   A list of registered APIs appears in the catalog.
3. In the list, click the API for which you want to apply the policy template.
5. On the Policy Editor screen, choose Edit -> Policy Template -> Apply.
6. Select the policy templates you want to apply from the Apply Policy Template window.
   You can click a template to view all the policies available and also select the required policies in that template.
7. If you want to copy only the policies and not the flows, choose Copy Only. Otherwise, choose Apply to copy both policies and flows.

Related Information

Create a Policy Template [page 453]
1.5.10.11.2 Update a Policy Template

Update a policy template.

Prerequisites

- You have a thorough understanding on Policies and the various Flows it can be attached to. For more information, see Policies [page 92].
- You are familiar with the different types of policies supported by SAP API Management. For more information, see Policy Types [page 93].

Context

You are updating a policy template.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
   A list of registered APIs appears in the catalog.
3. In the list, choose the API for which you want to update the policy template.
5. In the policy editor screen, choose Policy Template Update.
6. In the Update Policy Template window:
   - Select the policy template that you want to update, from the Name dropdown box.
   - Choose the required policies from the list of policies available.
7. Choose OK.

Related Information

Create a Policy Template [page 453]
Delete a Policy Template [page 458]
1.5.10.11.3 Import a Policy Template

Import a policy template.

Prerequisites

- You have a thorough understanding of policies and the various flows they can be attached to. For more information, see Policies [page 92].
- You are familiar with the different types of policies supported by API Management. For more information, see Policy Types [page 93].

Context

To import a policy template, proceed as follows:

Procedure

1. Log on to the API portal.
2. From the navigation bar, choose Develop. A list of registered APIs appears in the catalog.
3. On the Develop screen, choose POLICY TEMPLATES. A list of available policy templates appears in the catalog.
4. Choose Import
5. In the Import Policy Template window, attach the policy template you want to import and choose OK.

Related Information

Create a Policy Template [page 453]
1.5.10.11.4 Export a Policy Template

Export a policy template.

Prerequisites

- You have a thorough understanding of policies and the various flows they can be attached to. For more information, see Policies [page 92].
- You are familiar with the different types of policies supported by SAP API Management. For more information, see Policy Types [page 93].

Context

To export a policy template, proceed as follows:

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop. A list of registered APIs appears in the catalog.
3. On the Develop screen, choose POLICY TEMPLATES. A list of available policy templates appears in the catalog.
4. In the Actions column, select Export by choosing the icon for the policy template you want to export.

Related Information

Create a Policy Template [page 453]
Import a Policy Template [page 456]
1.5.10.11.5 Delete a Policy Template

Delete a policy template.

Prerequisites

- You have a thorough understanding of policies and the various flows they can be attached to. For more information, see Policies [page 92].
- You are familiar with the different types of policies supported by SAP API Management. For more information, see Policy Types [page 93].

Context

To delete a policy template, proceed as follows:

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
   A list of registered APIs appears in the catalog.
3. On the Develop screen, choose POLICY TEMPLATES.
   A list of available policy templates appears in the catalog.
4. In the Actions column, select Delete by choosing the icon for the policy template you want to delete.
5. Choose OK.

Related Information

Create a Policy Template [page 453]
1.5.10.11.6  Policy Template Structure

During an import or export of a policy template, the policy template follows a pre-defined structure.

<table>
<thead>
<tr>
<th>Folder Name</th>
<th>Path</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy Template Container</td>
<td>\PolicyTemplateContainer</td>
<td>Root folder that contains the FileResource and Policy information.</td>
</tr>
</tbody>
</table>
| FileResource               | \PolicyTemplateContainer\FileResource | Lists all the scripts attached to the policy. Only Java, Python, and XSL Scripts are supported. Follow the below naming convention:  
  ● Java Script: `<JavaScript name>.js`  
  ● Python script: `<PythonScript name>.py`  
  ● XSL script: `<XSLScript name>.xsl` |
| Policy                      | \PolicyTemplateContainer\Policy | Contains a list of all available policies. Each policy is available as a separate file with the naming convention `<Policy name>.xml`. |
| `<policytemplatename>.xml` | \PolicyTemplateContainer\<policytemplate>.xml | Contains the header information of all the available policies. |

1.5.11  Edit an API Proxy

Once you’ve created an API proxy you can further change the proxy, either on the API portal, or by using the embedded API designer.

Context

When you edit an API proxy either on the API portal or using the API designer, ensure that you save and deploy the API proxy once the changes are made. Saving the API proxy is a design time activity, the changes you’ve made get pushed to the runtime only when you deploy the changes. Therefore, when you choose Save after making the changes, the changes are saved locally and don’t get published on the API business hub enterprise. Choose Deploy to perform an explicit deployment to bring in the new changes in the runtime during the API proxy execution.

Consider the following examples:

- If you just save and not deploy the change you’ve made to the Target Endpoint of an API proxy, and then try to debug the API proxy and use the trace capability in runtime to trace the API call, the call points to the old Target Endpoint. Only when you save and then explicitly deploy the changes, the API call points to the new Target Endpoint.
**Procedure**

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose *Develop*.
3. Choose the API you want to edit.
   The *View API* page is displayed. To edit the various tabs available on this screen, choose *Edit* from the top-right corner of the screen.
4. Select the appropriate tabs, to edit the API. You can choose from the following, *Overview*, *Proxy Endpoint*, *Target Endpoint*, and *Resources*.

<table>
<thead>
<tr>
<th>Tab</th>
<th>Details</th>
</tr>
</thead>
</table>
| **Overview**     | You can edit the following:  
|                  | ○ Name of the API  
|                  | ○ Host Alias  
|                  | ○ API Base Path  
|                  | ○ API State  
|                  | ○ API Description |
| **Proxy Endpoint** | You can add the proxy endpoint and the route rules. |
| **Target Endpoint** | You can choose URL, API Provider, or API proxy, as the target endpoint as well as enter target endpoint rules. |
| **Resources**    | You can add resources, or change already existing ones. |
5. To edit the API in the embedded API designer, you can either choose **Edit > Edit in API Designer** from the top-right corner of the screen, or choose **Edit > Resources** tab, click > to open the API designer.

6. You can make required changes to the swagger structure in the API designer. For more information on the API designer, see *Creating APIs* [page 406].

7. Once you’ve made the swagger changes, click **Save**. These changes will then be reflected in the various tabs on the API portal.

   **Note**  
   When you’re editing the swagger structure in the API designer, editing the same from the tabs is disabled.

If the API proxy is already in the deployed state, then saving the changes after editing the API doesn’t deploy the latest changes. Similarly, if the API proxy is already published on the API business hub enterprise, the save action doesn’t publish the latest changes. In both cases, a message appears on the View API page that the changes you’ve made aren’t deployed. For the changes to reflect during API proxy runtime flow, choose **Click to Deploy** and provide your confirmation on the popup window.

**Results**

The changes you’ve made to the API are saved and deployed successfully.

**1.5.1.12 Key Value Map**

A key value map lets you create and manage collections of arbitrary key value pairs for any number of API proxies. Each key value pair is stored in a map as an entry.

**Note**  
It’s recommended that you avoid making any concurrent inserts and updates to the same key value map (KVM) scoped to the environment level as it may cause loss of data.

As a workaround, use API proxy scoped key value map.

**Caution**  
Don’t use Key Value Maps to store your logs as this can impact API proxy runtime flow. Instead, use the message logging policy to write your logs to external endpoints.

- Create a Key Value Map [page 462]  
  Create a key value map using the create and manage collections of arbitrary key value pairs for any number of API proxies.

- Update a Key Value Map [page 463]  
  Update a key value map.

- Delete a Key Value Map [page 464]
Delete a key value map which is not in use.

1.5.1.12.1 Create a Key Value Map

Create a key value map using the create and manage collections of arbitrary key value pairs for any number of API proxies.

Prerequisites

You are assigned the admin role.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Configure.
3. Choose Key Value Maps and choose Create.
4. On the Create screen, provide the following details:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Key value map name</td>
</tr>
<tr>
<td>Encrypted</td>
<td>Select this checkbox if you want to encrypt the values.</td>
</tr>
<tr>
<td>Key</td>
<td>Key name</td>
</tr>
<tr>
<td>Value</td>
<td>Value for the key</td>
</tr>
</tbody>
</table>

5. Choose Save.

Related Information

Delete a Key Value Map [page 464]
Update a Key Value Map [page 463]
1.5.12.2 Update a Key Value Map

Update a key value map.

Prerequisites

- You are assigned the admin role.
- You have created a key value map.

Context

You are updating a key value map to either add an entry, delete an entry, or update the value for an existing entry.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Configure.
3. Choose Key Value Maps and select the key value map that you want to edit.
4. Choose Edit to perform the following:
   a. If you want to add an entry, choose Add and provide the Key and Value.
   b. If you want to delete an entry, choose the delete icon in the Action column. Save the changes.
   c. If you want to update an entry, select the required entry and update the Value field.

   \[i\] Note
   You can only update the Value field and not the Key field.

5. Choose Save.

You can view the updated key value map by navigating to API Portal home page > Configure > Key Value Map.

Related Information

Create a Key Value Map [page 462]
Delete a Key Value Map [page 464]
1.5.12.3 Delete a Key Value Map

Delete a key value map which is not in use.

Prerequisites

You are assigned the admin role.

Context

You are deleting a key value map.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Configure.
3. Choose Key Value Maps and select the key value map that you want to delete.
4. In the Actions column, choose the delete icon for the key value map.

Related Information

Create a Key Value Map [page 462]
Update a Key Value Map [page 463]

1.5.13 Test APIs

Use the API Test Console to test the runtime behavior of APIs

SAP API Management provides an API Test Console, which enables you to test your APIs. Testing an API is essential to understand the runtime behavior of the APIs. The test console allows you to explore the resources associated with an API and execute the operations.

The API Test Console allows you to test OData and REST-based services.
**Procedure**

**Context:** You have logged on to the API portal or API business hub enterprise.

1. Log on to the API portal.
2. From the navigation bar on the left choose the ? **API Test Console**.
3. A list of APIs appears on the left.

   **Note**
   
   An API Admin can view both registered and published APIs. However, an App Developer can only view a list of published APIs.

4. Select the required API.
   The URL for the selected API is populated automatically in the **API Test Console**. For the selected API, the URLs of the supported resources appear in the dropdown list. One resource is selected by default.

5. If you want to choose a different collection, use the dropdown list to select the required collection.

6. If you have the URL of the service that contains the API, enter the service URL.

7. Choose **Authentication** to select the required type of authentication. You can choose from the following options:
   1. **None**: No authentication required.
   2. **Basic Authentication**: Provide a user name and password.

8. Enable the required method:
   1. **GET**: Reads an entity
   2. **POST**: Creates an entity
   3. **PUT**: Updates an entity
   4. **DELETE**: Deletes an entity

   **Note**
   
   You can enable only the methods supported by the service.

9. Enter the **Request Body** for PUT and POST methods.

10. Choose **Header** to add a header.

   **Note**
   
   If you want to add multiple headers, choose **Add Request Headers**.

11. Choose **Url Params** to enter the query parameter and value.

   **Note**
   
   If you want to add multiple query parameters, choose the button **Add URL Params**. Test Console supports passing of custom headers such as X-sap-apimgt-proxy-host:-proxy-trai and X-sap-apimgt-proxy-port:- 8080.

12. Choose **Send**.
   The response appears in the tabs:
   1. **Body**: View the formatted response.
   2. **Body (Raw)**: View the unformatted response.
   3. **Headers**: View the headers.
13. If you want to use the response body as an input request, choose Use as Request on the Body (Raw) tab.
14. To view the transactions based on the testing activity that you did, choose Launch API Viewer. For more information on tracing API proxy, see Debug API [page 466]

1.5.1.14 Debug API

You debug an API proxy to troubleshoot and monitor them in SAP API Management, by probing the details of each step through the API proxy flow.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. Select the required API that you want to debug.
5. Alternatively, you can also launch the debug viewer from the API Test Console by following the substeps below:
   a. Select the navigation icon and choose Test.
   b. Select the required API from the APIs list.
   c. Choose Debug at the bottom right corner.
6. In the API Debug Viewer, choose Start Debug.

When you start the debug, the API records details of each step in the processing pipeline. While the debug session is running, messages and contextual data are captured from live traffic.

i Note

One debug session supports 20 request/response transactions per message processor through the selected API proxy. A debug session automatically stops after 10 minutes if you don't manually stop it. You can also start the debug session at any point in time of working in the API Management.

You can view a list of captured request/response transactions on the left menu. Choose any of the transactions to view the detailed debug map and the corresponding properties.

7. To view the transaction details at any point in time of an active debug session, choose Refresh.
8. When you’ve captured a sufficient number of requests, choose Stop Debug.
9. Debug the API using the guidance provided below:

Transaction Map details
In addition to the above mentioned icons, each policy is represented by an icon. By choosing the icon, you can view the policy details.

**Phase Details**

Using phase details you can check the headers that are being sent to the backend, variables set by policies and so on. You can verify the base path to ensure that a policy is routing the message to correct server. Refer the table below to understand each phase:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy Endpoint</td>
<td>Indicates the selected proxy Endpoint flow for execution. An API proxy can have multiple named proxy endpoints</td>
</tr>
<tr>
<td>Request Headers</td>
<td>Lists the HTTP request headers</td>
</tr>
<tr>
<td>Request Content</td>
<td>Displays the HTTP request body</td>
</tr>
<tr>
<td>Phase</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Variables Read</td>
<td>Lists the flow variables that were read by a policy</td>
</tr>
<tr>
<td>Variables Read and Assigned</td>
<td>Lists the flow variables that were read and assigned a value by a policy.</td>
</tr>
<tr>
<td>Target Endpoint</td>
<td>Indicates the selected Target Endpoint for execution</td>
</tr>
<tr>
<td>Response Headers</td>
<td>Lists the HTTP response headers</td>
</tr>
<tr>
<td>Response Content</td>
<td>Displays the HTTP response body</td>
</tr>
</tbody>
</table>

### 1.5.1.15 Custom Attributes

This topic describes custom attributes and services. It is also used to create, delete, and update custom attributes for application and product entities.

Custom attributes can be leveraged to influence the runtime behavior of the API proxy execution. It can be set at a product level or at an application level (when application is created by admin on behalf of developer). Custom attributes provide the flexibility to extend the functionality based on attribute value which can be set or read during the API proxy execution flow. These attributes can be accessed during an API call via the following policies: Verify API key, Access token, and Access entity.

For example, if you add attributes for your products, applications and use Verify API key or Access token verification policy in your flow variables, this can enforce any kind of runtime limitations and control functions.

### Personas

<table>
<thead>
<tr>
<th>Role</th>
<th>Component</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthGroup.ContentAuthor</td>
<td>Developer portal</td>
<td>User assigned with this role can read, create, delete, and update custom attributes for application.</td>
</tr>
<tr>
<td>APIPortal.Service.Catalog.Integration</td>
<td>API Portal</td>
<td>User assigned with this role can read, create, delete, and update custom attributes for application.</td>
</tr>
<tr>
<td>APIPortal.Administrator</td>
<td>API Portal</td>
<td>User assigned with this role can read, create, delete, and update custom attributes for products</td>
</tr>
</tbody>
</table>
Limits

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom attribute name size</td>
<td>255 characters</td>
</tr>
<tr>
<td>Custom attribute value size</td>
<td>1024 characters</td>
</tr>
<tr>
<td>Number of custom attributes</td>
<td>18</td>
</tr>
</tbody>
</table>

Sample Payload

Sample payload to create a custom attribute

- **Url**: https://<consumer API-Portal host>:<port>/apiportal/api/1.0/Management.svc/APIProducts HTTP/1.1
- **Method**: POST
- **Content type**: application/JSON
- If you are in the Neo environment, fetch the x-csrf-token:
  - Service url: https://<consumer API-Portal host>:<port>/apiportal/api/1.0/Management.svc/APIProducts HTTP/1.1
  - **Method**: HEAD
  - Request Header: x-csrf-token: fetch
  - Response: x-csrf-token value.
- If you are in the Cloud Foundry environment, fetch the bearer token:
  - Service url: https://<consumer API-Portal host>/apiportal/api/1.0/Management.svc/APIProducts HTTP/1.1
  - **Method**: HEAD
  - Request Header: Authorization:Bearer <Token for API access>
  - Response: bearer-token value

To know how to retrieve this token, see API Access plan for API Portal [page 56].

Sample Code

```json
{
  "name": "SampleProduct",
  "version": "1",
  "isPublished": false,
  "status_code": "PUBLISHED",
  "title": "SampleProduct",
  "description": "SampleProduct",
  "isRestricted": false,
  "scope": "",
  "quotaCount": null,
  "quotaInterval": null,
  "quotaTimeUnit": null,
  "additionalProperties": [
    {
      "entityId": "SampleProduct",
      "name": "key1",
      "value": "val1"
    }
  ]
}
```
{

}

"entityId": "SampleProduct",
"name": "key2",
"value": "val2"

}
],
"apiProxies": [
{
"__metadata": {
"uri": "APIProxies(name='SampleAPI')"
}
}
],
"apiResources": [],
"__metadata": {
"type": "apiportal.APIProduct"
}

Sample payload to create a custom attribute (batch call)
 Sample Code
Content-Type: application/http
Content-Transfer-Encoding: binary
PUT APIProducts(name='SampleProduct') HTTP/1.1
Request Header: x-csrf-token: <value>
Accept-Language: en-US
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
Content-Type: application/json
Content-Length: 234
{"name":"SampleProduct","title":"SampleProduct","scope":"","description":"<p>S
ampleProduct</
p>","version":"1","status_code":"PUBLISHED","isRestricted":false,"isPublished"
:true,"quotaCount":-99,"quotaInterval":-99,"quotaTimeUnit":null}
--changeset
Content-Type: application/http
Content-Transfer-Encoding: binary
POST APIProductAdditionalProperties HTTP/1.1
x-csrf-token: fetch
Accept-Language: en-US
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
Content-Type: application/json
Content-Length: 60
{"entityId": "SampleProduct","name": "key3","value": "val3"}
Sample payload to update a custom attribute (batch call)
 Sample Code
Content-Type: multipart/mixed; boundary=changeset_9c02-68be-2f72
--changeset
Content-Type: application/http
Content-Transfer-Encoding: binary
PUT APIProducts(name='SampleProduct') HTTP/1.1
Request Header: x-csrf-token: <value>
Accept-Language: en-US
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0

470

PUBLIC

SAP API Management
SAP API Management in the Cloud Foundry Environment


Sample payload to delete a custom attribute (batch call)

- **Url:** https://<consumer API-Portal host>:<port>/apiportal/api/1.0/Management.svc/$batch HTTP/1.1
- **Method:** POST
- **Content type:** application/JSON
- **Request Header:** x-csrf-token: fetch (for Neo environment)
  
  Request header: Authorization: Bearer <Token for API access> (for Cloud Foundry environment)
  
  To know how to retrieve this token, see API Access plan for API Portal [page 56].

---

**Sample Code**

```json
Content-Type: multipart/mixed; boundary=changeset_9c02-68be-2f72
--changeset
Content-Type: application/http
PUT APIProducts(name='SampleProduct') HTTP/1.1
x-csrf-token: <value>
Accept-Language: en-US
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
Content-Type: application/json
Content-Length: 234
{"name":"SampleProduct","title":"SampleProduct","scope":"","description":"<p>SampleProduct</p>","version":"1","status_code":"PUBLISHED","isRestricted":false,"isPublished":true,"quotaCount":-99,"quotaInterval":-99,"quotaTimeUnit":null}
--changeset
Content-Type: application/http
Content-Transfer-Encoding: binary
PUT APIProductAdditionalProperties(entityId='SampleProduct',name='key3') HTTP/1.1
x-csrf-token: <value>
Accept-Language: en-US
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
Content-Type: application/json
Content-Length: 14
{"value":"vx"}
```
Sample payload to create a custom attribute (application)

- **Url**: https://<consumer Dev-Portal host>:<port>/odata/1.0/data.svc/APImgmt.Applications HTTP/1.1
- **Method**: POST
- **Content type**: application/JSON
- **If you are in the Neo environment, fetch the x-csrf-token**:
  - Service url: https://<consumer Dev-Portal host>:<port>/odata/1.0/data.svc/APImgmt.Applications HTTP/1.1
  - Method: HEAD
  - Request Header: x-csrf-token: fetch
  - Response: x-csrf-token value
- **If you are in the Cloud Foundry environment, fetch the bearer token**:
  - Service url: https://<consumer Dev-Portal host>/odata/1.0/data.svc/APImgmt.Applications HTTP/1.1
  - Method: HEAD
  - Request Header: Authorization:Bearer <Token for API access>
  - Response: bearer-token value

To know how to retrieve this token, see API Access Plan for API business hub enterprise [page 59].

### Sample Code

```json
{
  "id": "00000000000000000000000000000000",
  "version": "1",
  "title": "<AppName>",
  "developer_id": "b",
  "ToSubscriptions": [
    {
      "ToAPIProduct": [
        {
          "__metadata": {
            "uri": "APImgmt.APIProducts('<ProdName>')"
          }
        }
      ],
      "id": "00000000000000000000000000000000"
    }
  ],
  "ToAttributes": [
    {
      "name": "<AttributeName>",
      "value": "<AttributeValue>",
      "entityType": "Applications",
      "entityId": "00000000"
    },
    {
      "name": "<AttributeName>",
      "value": "<AttributeValue>",
      "entityType": "Applications",
      "entityId": "00000000"
    },
    {
      "name": "<AttributeName>",
      "value": "<AttributeValue>",
      "entityType": "Applications",
      "entityId": "00000000"
    }
  ]
}
```

Sample payload to create a custom attribute via navigation (application)
- URL: `https://<consumer Dev-Portal host>:<port>/odata/1.0/data.svc/`  
  `APIMgmt.Applications(<application_id>)/ToAttributes`

- Method: POST
- Content type: application/JSON

- If you are in the Neo environment, fetch the x-csrf-token:
  - Service URL: `https://<consumer Dev-Portal host>:<port>/odata/1.0/data.svc/`  
    `APIMgmt.Applications(<application_id>)/ToAttributes`
  - Method: HEAD
  - Request Header: `x-csrf-token: fetch`
  - Response: `x-csrf-token value`

- If you are in the Cloud Foundry environment, fetch the bearer token:
  - Service URL: `https://<consumer Dev-Portal host>/odata/1.0/data.svc/`  
    `APIMgmt.Applications(<application_id>)/ToAttributes`
  - Method: HEAD
  - Request Header: `Authorization: Bearer <Token for API access>`
  - Response: `bearer-token value`

To know how to retrieve this token, see [API Access Plan for API business hub enterprise](page 59).

### Sample Code
```
{
    "name": "<AttributeName>",
    "value": "<AttributeValue>",
    "entityType": "Applications",
    "entityId": "0000000"
}
```

Sample payload to update a custom attribute (application)

- URL: `https://<consumer Dev-Portal host>:<port>/odata/1.0/data.svc/`  
  `APIMgmt.Attributes(name=<attribute_name>,entityId=<application_id>,entityType='Applications')`

- Method: PUT
- Content type: application/JSON

- Request Header: `x-csrf-token: fetch`

### Sample Code
```
{
    "name": "<AttributeName>",
    "value": "<AttributeValue_updated>",
    "entityType": "Applications",
    "entityId": "0000000"
}
```

Sample URL to delete a custom attribute (application)

- URL: `https://<consumer Dev-Portal host>:<port>/odata/1.0/data.svc/`  
  `APIMgmt.Attributes(name=<attribute_name>,entityId=<application_id>,entityType='Applications')`

- Method: DELETE
- Content type: application/JSON
Add Custom Attributes to a Product

Add custom attributes to a product.

Prerequisites

- You are assigned the admin role.

Context

Use this procedure to add custom attributes to a product.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and navigate to Develop.
3. Choose Products and select the product for which you want to add the custom attribute.
4. In the product details page, choose Custom attributes.
5. In the Custom attributes section, choose Add. Provide a Name and Value for the custom attribute. To add more attributes, choose Add:
   - To delete a custom attribute, choose the delete icon under the Actions column.
6. Save the changes.
1.5.1.15.2 Restoring Application ID across SAP API Management Landscapes

Maintaining the same application ID across SAP API Management landscapes.

Prerequisites

- You have the AuthGroup.API.Admin role assigned to you.
- You have the application ID for the application you want to port or recreate.

**Note**
The application ID only contains alphanumeric characters, underscores(_), and hyphens(-).

Context

Each application created in a particular SAP API Management landscape is associated with an application ID. When you are migrating to a new SAP API Management landscape, you can recreate the application there, but this will be associated with a new application ID. To ensure that identical applications are available across SAP API Management landscapes, the original application ID is used via an API provided in this document.

Procedure

- Use the POST operation on the following service URL to create an application. The request body to be used for the POST operation is provided in the sample code.

  **URL**
  
  `<dev-portal-host>/odata/1.0/data.svc/APIMgmt.Applications`
  
  - The host changes in the above URL depending on what is used.

  **Code Syntax**
  
  ```
  {
    "id": "<application_id>",
    "version": "1",
    "title": "<application_title>",
    "description": "<application_description>",
    "callbackurl": null,
    "developer_id": "<developer_id>",
    "ToSubscriptions": [ ]
  }
  ```
Using this API will ensure that applications are recreated or ported to the new SAP API Management landscape with the same application ID and are identical.

1.5.1.16 Load Balancing Across API Providers

Load-balancing is configured to distribute the load efficiently across multiple API providers.

Load-Balancing can be applied to an API proxy only when the API proxy is created with a link to an API provider. For more information on how to link an API proxy to the API provider, refer to the Create an API [page 393].

To perform all the operations related to the target endpoint, for example, fetching the resources and synchronizing all the operations, the API proxy is linked to an API provider. For load balancing, additional API providers are linked to the API proxy.

**Note**
All the API providers that are linked to the API proxy must exist in design time.

Configure Load Balancing During Import [page 476]
You can apply load-balancing functionality to an API proxy, by including the load balancer, and health monitor attributes in a .zip file along with the API proxy content.

Configuring Load Balancing [page 480]
You can configure load-balancing functionality for an API proxy from the API Management, API Portal.

1.5.1.16.1 Configure Load Balancing During Import

You can apply load-balancing functionality to an API proxy, by including the load balancer, and health monitor attributes in a .zip file along with the API proxy content.

You can attach the .zip file while importing an existing API definition in to the API portal. For more information, refer Import an API [page 448].

You can configure load balancer and health monitor by adding the below attributes to \APIProxy \APITargetEndPoint\default.xml in the design time .zip file as shown in the following example:
The load-balancing attributes in the sample code are defined in the table below:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxFailures</td>
<td>Specifies the number of failed requests from the API proxy to the API provider that results in the request being redirected to another API provider. You must set &lt;MaxFailures&gt; greater than 0 when using the HealthMonitor. When &lt;MaxFailures&gt; value is configured as 0, the Load Balancer tries to connect to the API provider for each request and never removes the API provider from the rotation.</td>
</tr>
<tr>
<td>fallBackServer</td>
<td>When all the additional providers fail, then all the requests are sent to this fallback server. When the load balancer determines that all API providers are unavailable, all traffic is routed to the fallback server.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Definitions</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Algorithm</td>
<td>By default <strong>RoundRobin</strong> algorithm is used. But you can also use <strong>Weighted</strong> and <strong>Least Connection</strong> algorithms.</td>
</tr>
<tr>
<td></td>
<td>The round robin algorithm forwards a request to each API provider in the order in which the API providers are listed in the target endpoint HTTP connection.</td>
</tr>
<tr>
<td></td>
<td>The <strong>Weighted</strong> load-balancing algorithm enables you to configure proportional traffic loads for your API providers. The weighted load-balancer distributes request to your API providers in direct proportion to each API provider’s weight. Therefore, the weighted algorithm requires you to set a weight attribute for each API provider as shown in the example below:</td>
</tr>
<tr>
<td></td>
<td>In this example, two requests are routed to API providers for every request routed to <code>&lt;provider_id&gt;target2&lt;/provider_id&gt;</code>.</td>
</tr>
<tr>
<td></td>
<td>You can also configure the load-balancer to use the <strong>Least Connection</strong> algorithm. This algorithm routes outbound requests to the API providers with fewest open HTTP connections.</td>
</tr>
<tr>
<td>serverUnhealthyResponseCode</td>
<td>This attribute is added to help ensure that bad HTTP responses, such as 500, increments the failure counter to take an unhealthy server out of load-balancing rotation as soon as possible.</td>
</tr>
</tbody>
</table>

**Sample Code**

```xml
<additionalAPIProviders>
  <provider_id>target1</provider_id>
  <provider_id>target2</provider_id>
  <provider_id>target3</provider_id>
</additionalAPIProviders>

<loadBalancerConfigurations>
  <algorithm>Weighted</algorithm>
  <isRetry>false</isRetry>
  <fallbackServer>target1</fallbackServer>
  <serverUnhealthyResponseCode>500, 502, 503</serverUnhealthyResponseCode>
  <weights>2</weights>
</loadBalancerConfigurations>
```

You can also configure the load-balancer to use the **Least Connection** algorithm. This algorithm routes outbound requests to the API providers with fewest open HTTP connections.
### Attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>isRetry</td>
<td>If retry is enabled, a request is sent whenever a response failure occurs, for example I/O error, or HTTP timeout. A request is also sent whenever the response received matches a value set by the <code>&lt;serverUnhealthyResponseCode&gt;</code></td>
</tr>
</tbody>
</table>

**HealthMonitor with HTTPMonitor/ TCPMonitor Configuration Attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>intervalInSec</td>
<td>The time interval, in seconds, between each polling TCP/HTTP request.</td>
</tr>
<tr>
<td>isEnabled</td>
<td>A boolean that enables or disables the health monitor.</td>
</tr>
<tr>
<td>connectTimeoutInSec</td>
<td>Time in which connection to the TCP/HTTP port must be established, to be considered a success. Failure to connect in the specified interval counts as a failure, incrementing the load balancer’s failure count for the API provider.</td>
</tr>
<tr>
<td>socketReadTimeoutInSec</td>
<td>Time, in seconds, in which data must be read from the HTTP service to be considered a success. Failure to read in the specified interval counts as a failure, incrementing the load balancer’s failure count for the API provider.</td>
</tr>
<tr>
<td>port</td>
<td>The port on which the HTTP connection to the API provider is established.</td>
</tr>
<tr>
<td>verb</td>
<td>Currently, only GET operation is supported. HTTPMonitor submits a GET request to the API provider.</td>
</tr>
<tr>
<td>path</td>
<td>The path appended to the URL defined in the API provider. Use this path element to configure a 'polling endpoint' on your HTTP service.</td>
</tr>
<tr>
<td>successResponse</td>
<td>Matching options for the inbound HTTP response message generated by the polled API provider. Responses that don’t match increment the failure count by 1.</td>
</tr>
<tr>
<td>responseCode</td>
<td>The HTTP response code expected to be received from the polled API provider.</td>
</tr>
</tbody>
</table>

**Parent topic:** [Load Balancing Across API Providers](#)

**Related Information**

[Configuring Load Balancing](#)
1.5.1.16.2 Configuring Load Balancing

You can configure load-balancing functionality for an API proxy from the API Management, API Portal.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
   A list of registered APIs appears in the catalog.
3. Browse for an API proxy, which is already linked to an API provider.
4. Choose the slide button to enable the Load-Balancing functionality.
5. Select additional API providers from the API Provider dropdown menu.
   The FallBack Server field appears once the additional API providers are selected.
   Select one of the additional API providers as the FallBack Server from the dropdown menu.
   When the load balancer determines that the additional API providers are unavailable, all traffic is routed to the fallback server.
6. Choose the slide button to enable Retry.
   If retry is enabled, a request is sent again whenever a response failure occurs, for example I/O error, or HTTP timeout. A request is also sent whenever the response received matches a value set by the Response Code.
7. By default Round Robin algorithm is selected. But you can also select Weighted and Least Connection algorithms.
   The Round Robin algorithm forwards a request to each API provider in the order in which the API providers are listed in the target endpoint HTTP connection.
   The Weighted load-balancing algorithm enables you to configure proportional traffic loads for your API providers. The weighted load-balancer distributes request to your API providers in direct proportion to each API provider’s weight. Therefore, the weighted algorithm requires you to set a weight attribute for each API provider.
   You can also configure the load-balancer to use the Least Connection algorithm. This algorithm routes outbound requests to the API providers with fewest open HTTP connections.
8. Choose the slide button to enable Maximum Failure.
   When enabled, it checks for the number of failed requests from the API proxy to the API provider that results in the request being redirected to another API provider.
   Maximum Failure Value: Enter the maximum number of failed requests from the API proxy to the API provider. Set the maximum failure value greater than 0 when using the Health Monitor. When the value is configured as 0, the Load Balancer tries to connect to the API provider for each request and never removes the API provider from the rotation.
i Note
The provider is removed from load balancer configuration, if the maximum number of failed requests is reached.

Response Code: Enter the HTTP response codes that are expected to be received from the polled API providers.

9. Choose Save.

Results

You have configured load balancing for the API proxy.

Task overview: Load Balancing Across API Providers [page 476]

Related Information

Configure Load Balancing During Import [page 476]

1.5.2 Publish APIs

You need to publish APIs to make it consumable by external application developers. Publishing enables to expose the APIs in a structured manner in the form of a product. To publish APIs, you need to know how to bundle APIs and publish them together as a product.

A product is a bundle of APIs. It contains metadata specific to your business for monitoring or analytics. For example, all APIs related to CRM can be bundled as one CRM product. Instead of publishing APIs individually, it is easier to bundle related APIs together as a product and publish it. After including the required APIs to a product, the product is published to the Catalog, where the product is available for Application developers to browse through.
When you create a product, you link it to one or more APIs. Also, the same API can be linked to multiple products. After you have linked an API to a product, all attributes of the API such as API resources and API documentation are implicitly part of the product.

A product is a vehicle that lets the Application developer know which APIs are exposed on the API portal. When you create an application, you select the product to include in the application. For each application that you create, SAP API Management generates an Application key and secret. Use this key to gain access to multiple products.

1.5.2.1 Create a Product

Explains how to create products to publish a bundle of APIs together.

You create a product when you want to expose one or more APIs to the Application Developer.

Prerequisites

You’ve created the required API on the APIs tab. For more information about how to create APIs, see Create an API Proxy [page 390].

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. Go to the Products tab. A list of published products appears.
   You can view the number of calls made for all APIs in a product for the current month. The data is visible for each product in the Calls column and also on the details screen of the individual product. You can click the refresh icon to get the latest data.
   
   The data is displayed according to metric specifications, for example:
   - 999 shows as 999 and 1000 shows as 1K
   - 999000 shows as 999K and 1000000 shows as 1M
   - 1500000 shows as 1.5M and 1000000000 shows as 1G
4. To create a product, choose Create.
5. Enter a name, title, and description for the product.
6. Specify the quota limits for this product.
**Note**

To enforce a quota on products, you must define verify API key and quota policies on the API. Setting quota limits on a product doesn’t automatically enforce a quota on the API proxies. The quota set on the product takes precedence over that of the API proxy. It’s a default limit that is referenced in quota policies that stipulate a uniform setting across all API proxies in the product. You can make runtime changes to the quota setting on an API product, and quota policies that reference the value automatically are updated with the new quota. For more information, see Quota [page 217].

You can use the sample payload given below to set **Verify API Key policy** for the required API:

```xml
<!-- Specify in the APIKey element where to look for the variable containing the api key -->
<VerifyAPIKey async='true' continueOnError='false' enabled='true' xmlns='http://www.sap.com/apimgmt'>
  <APIKey ref='request.header.apikey'/>
</VerifyAPIKey>
```

You can use the sample payload below on the same API to create **Quota policy**:

```xml
<!-- can be used to configure the number of request messages that an app is allowed to submit to an API over a course of unit time -->
<Quota async="false" continueOnError="false" enabled="true" type="calendar" xmlns="http://www.sap.com/apimgmt">
  <Identifier ref='verifyapikey.vap1.client_id'/>
  <!-- specifies the number of requests allowed for the API Proxy -->
  <Allow countRef="verifyapikey.vap1.apiproduct.developer.quota.limit" count="100"/>
  <!-- the interval of time for which the quota should be applied -->
  <Interval ref="verifyapikey.vap1.apiproduct.developer.quota.interval">1</Interval>
  <!-- used to specify if a central counter should be maintained and continuously synchronized across all message processors -->
  <Distributed>true</Distributed>
  <!-- Use to specify the date and time when the quota counter will begin counting, regardless of whether any requests have been received from any apps -->
  <StartTime>2015-11-11 12:00:00</StartTime>
  <!-- if set to true, the distributed quota counter is updated synchronously. This means that the update to the counter will be made at the same time the API call is quota-checked -->
  <Synchronous>true</Synchronous>
  <!-- Use to specify the unit of time applicable to the quota. Can be second, minute, hour, day, or month -->
  <TimeUnit ref="verifyapikey.vap1.apiproduct.developer.quota.timeunit">month</TimeUnit>
</Quota>
```

You’ll notice that the same API key is used in the quota policy. You can now update the policy.
7. To set the scope at product level to restrict the access of the authorization token for each application, specify the scope in the Scope field. For example, you can set the scope as Read-only, Read-write, and so on.

The OAuth 2.0 policy provides a way to limit the amount of access that is granted to an access token. For example, an access token issued to a client app may be granted READ and WRITE access to protected resources, or just READ access. You can implement your APIs to enforce any scope or combination of scopes you wish. So, if a client receives a token that has READ scope, and it tries to call an API endpoint that requires WRITE access, the call will fail.

Each product can have zero to many scopes assigned. These scopes can be assigned when the product is created or later. Scopes exist as a list of names and are included in the metadata associated with each product.

8. In the APIs section, choose Add.

9. In the Add APIs window, select the required APIs and the corresponding resources.

### Note

While selecting APIs and its resources for product creation, the following behaviours apply when API calls are made to the selected APIs and resources:

- Product creation in API Management provides precedence for product to path (resource) mapping over product to API mapping. Let’s understand this behavior with an example:
  - Let’s say you want to create a product P1 that consists of 2 APIs namely API_1 and API_2.
  - API_1 contains resources namely R1 and R2.
  - API_2 contains resources namely R2 and R3.
  - That is API_1=R1,R2 and API_2=R2,R3

As you can see, R2 is a common resource that exists in both the APIs. Now, for your product creation, let’s say you select resources R1, R2 of API_1 and resource R3 of API_2. Thus, your product consists of resources R1 and R2 from API_1 and R3 from API_2. That is P1=R1,R2,R3.

With the above resource selection criteria, API Management still allows API calls to be made to the resource R2 of API_2 even though you had not explicitly selected the resource under API_2 during product creation.

- API Management doesn’t support API calls to those resources whose path starts with a /$ character. That is, if you create a product by attaching individual resources, then API calls to those resources whose path starts with /$<resource_name> don’t work. However, when you attach the whole API and none of its resources to a product, then API calls made to those resources of the selected API still works irrespective of whether the path starts with /$ character or not.

### Note

Select at least 1 API to publish a product.

### Note

Make sure that the API is deployed before attaching it to a Product. If you try to publish a Product that has an API with saved changes attached to it, the following error message appears: "The API proxy attach to the Product has some changes that aren’t deployed yet.”

Similarly, if the Product has multiple APIs attached to it, and few of the APIs have changes that are saved but not deployed, you’ll receive the following message when you try to publish the Product: "The following API Proxies attached to the Product weren’t published as they have changes that aren’t yet deployed:”
10. Choose OK. The selected APIs are listed on the APIs tab.

11. Provide permissions to user roles to either discover or subscribe to the product.

12. In the Rate plans section, choose Add.

13. In the Add Rate Plan window, select the rate plans that you want to add to this product and choose OK.

14. (Optional) In the Custom Attribute section, specify the custom attributes you want to add to the product. A custom attribute is a name/value pair, which can be used in multiple ways, including influencing the runtime behavior of an API proxy. Custom attributes provide the flexibility to extend the functionality based on attribute value, which can be set or read during the API proxy execution flow. These attributes can be accessed during an API call via the following policies: Verify API key, Access token, and Access entity.

  **Note**
  You can add a maximum of 18 custom attributes.

For example, you can create a custom attribute named IsConfidential with a value of Yes or No. Later, in your API proxy flow, you can check the value of the API product’s IsConfidential attribute (for example, using the verifyapikey.<policy_name>.apiproduct.IsConfidential variable, which would be available automatically after you have created the custom attribute). If the value is Yes, you can throw an error, for example as shown below using the Raise Fault policy.

  **Sample Code**
  Warning: You do not have relevant authorizations to access the information.

  **Note**
  The verifyapikey.<policy_name>.apiproduct.IsConfidential would be available only if the Verify API Key policy in your API proxy is executed successfully.

You can use the following sample payload to set Verify API Key policy for the required API:

  **Sample Code**
  <!--Specify in the APIKey element where to look for the variable containing the api key-->
  <VerifyAPIKey async='true' continueOnError='false' enabled='true'
      xmlns='http://www.sap.com/apimgmt'>
    <APIKey ref='request.queryparam.apikey '/>
  </VerifyAPIKey>

You can use the following condition and sample payload to set the Raise Fault policy:

  **Sample Code**
  verifyapikey.Verify-API-Key.apiproduct.IsConfidential=True

  **Sample Code**
  <!-- can be used to create custom messages in case of an error condition -->
  <RaiseFault async="true" continueOnError="false" enabled="true"
      xmlns="http://www.sap.com/apimgmt">
15. Once you have filled in all the required details for the product, you can choose one of the following two actions:
   - **Save as Draft** - The resulting state of the product is Draft, and you can publish the product anytime. The product cannot be used for creating applications when it is in the Draft state.
   - **Publish** - The resulting state of the product is Published, and it is available for creating applications. You can edit the published product anytime.

16. If you want to delete a product, select the required product from the catalog and choose Delete. If it is being used in an application.

17. If you want to edit a product, select the required product, in the details view select **Edit**.

---

**Note**
Alternatively, you can use the following service to update the product:

```bash
--batch_1ddf-343f-3338
Content-Type: multipart/mixed; boundary=changeset_418f-1c1d-b76b
--changeset_418f-1c1d-b76b
Content-Type: application/http
Content-Transfer-Encoding: binary
PUT APIProducts(name='"<Product_Name>"') HTTP/1.1
Content-Type: application/json
Content-Length: <Length of below payload>
"name":"<Product_Name>"","title":"<Product_Title>"","scope":"","description":"<Product_Description>"","version":"1","status_code":"PUBLISHED","isRestricted":<true or false>,"isPublished":true,"quotaCount":-99,"quotaInterval":-99,"quotaTimeUnit":null
--changeset_418f-1c1d-b76b--
```
1.5.2.1.1 Assign Permission to a Product via UI

Assign permission to a product in the API portal.

Prerequisites

- You are assigned the APIPortal.Administrator role.
- You must have created a custom role on the SAP BTP Cockpit Cloud Foundry environment. For more information on creating a custom role, refer here [page 67].

Context

Whenever you create a product or edit a draft product, permissions can be added to product. Use this procedure to grant permission to user roles for discovering and subscribing to the product in the API business hub enterprise. Only users who are assigned the required role can discover and subscribe to the product.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. Go to the Products tab and choose Permission.
   
   Here, whenever a product is created or a draft product is edited, permissions can be added to product.
4. Select a role from the Discovery dropdown list. Only users who are assigned the selected role can discover this product in the API business hub enterprise.
5. Select a role from the Subscription dropdown list. Only users who are assigned the selected role can subscribe to this product in the API business hub enterprise.
   - You can change the roles selected for Discovery and Subscription by choosing Edit.
   - You can remove the roles selected for Discovery and Subscription by choosing Remove Role.

Related Information

Create a Product [page 482]
1.5.2.2 View Applications

Viewing subscribed applications from API portal.

Context

As an admin, you can now view the subscribed applications from API portal. You can view developer details and associated products and not AppKey and secret.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. Navigate to the Applications tab page.

A list of Applications appears. If you have logged on to the API portal with the role APIPortal/Administrator, then you can view the Application key and secret.

You can view the number of calls made for all APIs in an application for the current month. The data is visible for each application in the Calls column and also in the details screen of individual application.

You can click on the refresh icon to obtain the latest data.

i Note
There is some delay in reflecting the latest data.

Notion used to display the data is as per metric specifications, for example:
- 999 shows as 999 and 1000 shows as 1k
- 999000 shows as 999K and 1000000 shows as 1M
- 1500000 shows as 1.5M and 1000000000 shows as 1G

Related Information

Create an Application [page 511]

1.5.3 Analyze APIs

Use the capabilities of API Analytics to analyze API usage and performance.

API Management provides comprehensive analytics capabilities to understand the various patterns of API consumption and performance. The API Analytics server uses the runtime data of the APIs to analyze the
information. The runtime data is gathered, analyzed, and displayed as charts, headers, and key performance indicators (KPIs).

Use the analytics dashboard to view the aggregated results. The detailed analytics view helps to manage APIs, attract the right application developers, troubleshoot problems, and ultimately, make better business decisions.

There are two variants of analytics dashboard available for analyzing API reports:

- **API Analytics**
  The analytics dashboard provides a comprehensive view of analytical charts and KPIs relevant to the performance of the APIs. The dashboard also displays the error-related charts and KPIs such as total number of API errors, total number of system errors, and policy errors. For more information, see [API Analytics](#page489).

- **Advanced API Analytics**

  **Note**
  We are calling this new flavor of analytics, Advanced API Analytics. However, for you, as an end-user there will be no direct reference to this title on the user interface. We are sure that the various new features will surely enable you to make best use of this exciting new analytics dashboard.

  Advanced API Analytics brings to you the all new analytics dashboard, providing powerful tools and in-depth reports for analyzing your API usage and performance. The reports are categorized across report pages, with each report page providing information about key API metrics, which are relevant for both business users and API developers. For more information, see.

**Related Information**

[Analytics Dashboard](#page490)
[Find Your Way around Advanced API Analytics Dashboard](#page493)

### 1.5.3.1 API Analytics

API Analytics provides sample analytical charts and key performance indicators (KPIs). These charts and KPIs are preconfigured in the dashboard.

Examples of standard analytical charts include:

- What is the API traffic trending over time?
- Which five APIs have the slowest response time?
- What is the overall API error count?

Examples of standard KPIs include:

- Total API hits
- API response time
• Total number of policy errors

Related Information

Analytics Dashboard [page 490]
Working with the Analytics Dashboard [page 490]

1.5.3.1.1 Analytics Dashboard

The analytics dashboard has some common features such as the views you can choose, the time range for which you want to display data, resize charts, and so on.

Features of the Analytics Dashboard

The following lists common features on the dashboard:

1. **Views**: The dashboard provides three views:
   - **Performance View**: Displays the performance-related charts and KPIs, such as API traffic for a specific period of time.
   - **Error View**: Displays error-related charts and KPIs, such as total number of API errors.
   - **Custom View**: Displays custom charts that you created by selecting measures, dimension, and chart type.

2. **Time interval**: Displays data only for the selected period of time. For example, time interval of months, weeks, and so on.

3. **Resize charts**: Resize charts from small, medium to large.

Related Information

Working with the Analytics Dashboard [page 490]

1.5.3.1.2 Working with the Analytics Dashboard

The analytics dashboard provides a comprehensive view of API performance and errors in the form of charts and KPIs.

The analytics dashboard provides a comprehensive view of API performance and errors in the form of charts and KPIs. Use the dashboard to do the following:
• View the performance of all APIs
• View the errors of all APIs
• Filter data based on a selected period

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Analyze. The Analytics dashboard appears.
3. Use the dropdown list on the top left to select one of the following views:
   ○ Performance View: Displays performance-related charts and KPIs. For example, you can display a chart showing API traffic, or view performance-related KPIs such as total number of API hits, average response time for APIs, and so on.
   ○ Error View: Displays error-related charts and KPIs. For example, you can display a chart showing the top 5 API policy-related errors, or view error-related KPIs such as total error count, total number of system errors, total number of API policy errors, and so on.
   ○ Custom View: Displays custom charts that you have created by selecting measures, dimensions, and the chart type from the pane on the left.
4. For every chart on the dashboard, you can perform the following actions:
   ○ Details: Navigate to the details page of any chart by choosing Details in the top right corner of the chart. On the details page, you can switch between different chart types and apply filters.
   ○ Resize: Change the size of the chart (small, medium, or large).
5. If you want to analyze data for a selected period, choose the required time period at the top of the dashboard. You can view data for the following time periods:
   ○ Last day
   ○ Last 7 days
   ○ Last 30 days
   ○ Last 6 months
   ○ Custom range (date interval and frequency) according to your requirements
6. If you want to customize an analysis using particular measures and dimensions, perform the following substeps:
   1. Select Analyze Data in the bottom right corner of the screen.
   2. Enter a chart title of your choice in the Title field.
   3. Add the required measures and dimensions in the left pane.
   4. The selected measures appear in the Selected Measures dropdown list. You can select the required measure from this list and plot the chart.

   **Note**
   You can add multiple measures with various dimensions for a single chart. If you add more than two measures, a table with measure details is displayed below the chart.

5. To drill down on a particular dimension, click the corresponding bar on the plotted chart. If you apply a filter to a chart to drill down to the details, you can navigate back to any previous parameter by using the breadcrumb option.
6. If you want to analyze using a particular value of measure or dimension in the plotted chart, choose the Filter icon at the top of the chart and set the required values in the Filter popup. You can enter values...
for measures manually using theEquals to, Greater than, or Less than options. For static dimensions (default), you can choose multiple values from the dropdown list. For custom dimensions (created by you), you can enter multiple values separated by commas.

7. Choose **OK** and then save the chart.
8. The chart appears in the **Custom View**.
9. To remove a custom chart from custom view, select Unpin from Custom View option from the Chart Settings dropdown.
10. To add a custom chart, select Change Analytic View option from the Chart Settings dropdown.
11. To delete a custom chart, select Delete option from the Chart Settings dropdown.
12. To edit a custom chart, choose Edit Chart icon.

**i Note**

As you analyze your data, you may see an entity value of (n.a or not set) displayed, including the parentheses, for your API Proxies, Product, Developer, and Developer Apps dimensions. This means one of the following:
- Not all of your API proxies and developer applications are using products.
- Some of your traffic is generated by unregistered developers and applications. This traffic may originate from an internal-use or public API.

### 1.5.3.2 Advanced API Analytics

Advanced API Analytics brings to you the all new analytics dashboard, providing handy and powerful analytical reporting tools to track your API performance and usage.

**i Note**

We are calling this new flavor of analytics, Advanced API Analytics. However, for you, as an end-user there will be no direct reference to this title on the user interface. We are sure that the various new features will surely enable you to make best use of this exciting new analytics dashboard.

Most of the reports on the analytics dashboard are a graphical representation of data, derived using visually appealing charts. You can choose between different chart types to visualize data as per your needs. The chart-oriented representation enables you to quickly glance through and analyze important API metrics, thus helping you in making better business decisions. The analytical data is spread across various report pages namely Overview, Health, and Usage, with each page providing information about key API metrics.

- **Overview**

  The Overview page provides a concise report about important and key API metrics. In the Overview page, both business users and API developers can quickly analyze reports and view API trends for the last seven days. Business users can obtain information about most popular APIs, total number of API calls, and key application developers. Developers can obtain information about non-performing APIs and the factors affecting these APIs such as response time and latency.

- **Health**

  The Health page provides reports about key metrics related to the performance of your APIs. In the Health page, API developers can quickly monitor the API metrics that affect the performance of APIs and view API error trends for the last seven days. The key monitoring metrics include information about average API
response time and the common types of API errors. API developers can also obtain information about recent error responses for an API and the total number of erroneous calls.

- **Usage**
  The Usage page provides reports about key metrics related to user-engagement. In the Usage page, business users can analyze API metrics that indicate the overall traction or acceptance of their API program and view API trends for the last three months. The key user-engagement metrics include information about the sources or medium from where you are acquiring users and traffic to your APIs. Business users can also obtain information about new developers who are onboarded to their API program, and a list of recently created applications.

### Related Information

Find Your Way around Advanced API Analytics Dashboard [page 493]

#### 1.5.3.2.1 Find Your Way around Advanced API Analytics Dashboard

Familiarize yourself with the main features and controls of the Advanced API Analytics dashboard.

### Help and Notifications Menu

Every page in Analytics dashboard gives you access to notifications, help documentation, and lets you manage your SAP API Management account. This menu is available at the top-right corner of the analytics dashboard.

Click 🗣 to know about the latest news and updates for API Management.

Click ⚙️ to view the version of API Management, access the online help documentation, and logout of API portal.

### Report Pages

In the Analytics dashboard, you access all your reports in report pages. The reports are categorized into report pages namely Overview, Health, and Usage. These report pages provide information about key metrics related to your API usage and performance.

You can find a time zone switcher to the right side of the report pages. The time zone switcher allows you to view analytics data based on different time zones. The default time zone shown is UTC. The time zone switcher is available across all the report pages including custom report pages.
Note

The data retention period for all report types available in the analytical dashboard is 6 months. That is, the
analytical dashboard stores and retains data only for a period of 6 months. After the retention period, the
data is purged.

Overview

The Overview page provides a summarized report about your most important and key API metrics. By default,
the Overview page provides report data for the last seven days.

At the top of the Overview report page, the following key API metrics are represented in a tile format:

- **Total API Calls**: Total number of API requests made over a specified period.
- **API Response Time**: Total time in milliseconds to respond to an API request. This round-trip time includes
  the API proxy overhead and the target server time.
- **Request Processing Latency**: Total time in milliseconds from the time when a request reaches the
  selected API proxy to the time when the proxy sends the request to the target server.
- **Total API Errors**: Total number of errors that occur on the API proxy plus the errors that occurs on the
  target server from all the API requests and responses over a specified period.
- **Target System Errors**: Total number of errors that occur on the target server over a specified period.
- **Target Response Time**: Total time in milliseconds for the target server to respond to an API request.

Each tile shows weekly percentage difference in data for a key API metric. A green-arrowed percentage
difference indicates a healthy API metric, whereas, a red-arrowed percentage difference indicates the API
metric needs improvement.

Hopping over each tile shows data for the current week, last week, and the difference between current week
and last week.

Just below the key API metrics tiles, you can filter and check for the number of calls for each of your APIs using
the **API Calls** dropdown menu. By default, it shows the total number of success and failure calls for all the APIs,
combined.

The rest of the Overview page provides a graphical view of other key API metrics, which include the following:

- Top APIs of the Week
- Top Products of the Week
- Top Applications of the Week
- Top API Providers of the Week
- Top Developers of the Week
• Top APIs
• Top Applications
• Top Products
• Top API Providers
• Top Developers
• Top Response Codes
• API Errors
• API Response Time
• Developer Engagement
• Top Backend Errors
• Top Proxy Errors
• Slowest APIs

**i Note**

In any of the key API metrics, if you observe an entity in the chart marked as Unidentified, it indicates the following:

- **Unidentified Application**: An application could not be identified for these API calls as the API key could not be verified.
- **Unidentified Product**: A product could not be identified for these API calls as the API key could not be verified.
- **Unidentified Developer**: A developer could not be identified for these API calls as the API key could not be verified.

**Health**

The Health page provides reports about key metrics related to the performance of your APIs. By default, it shows data for the last seven days.

At the top of the Health page, there is a date-range selector. This date-range selector lets you set the time period for which you want to analyze the reports. To set a new time period, click and drag the bubble-like endpoints on the date-range selector.

Above the date-range selector, select **Day**, **Hour**, or **Minutes** tabs to see daily, hourly, or 30-minute aggregate data.

The **Day** option displays seven touch points, one for each day of the week.

The **Hour** option displays 24 touch points, one for each hour of the day.

The **Minutes** option displays 48 touch points, one for every 30 minutes of the day.

At the top right corner of the Health page, click-arrow to view advanced filter menu and options. The filter menu and options appear below the date-range selector and you can filter your reports by API, Applications, Products, or Developers. Once you apply the filter options, the applied filters are displayed under **Active Filters**.
The rest of the Health page displays a graphical view of key API metrics, which include the following:

- API Calls
- Response Code Count
- Cache Responses
- Backend Error Call Count
- Backend Response Time
- API Error Calls
- API Response Time
- Error Count per Response Code
- Average Response Time

**Usage**

The Usage report page provides reports on key metrics related to user-engagement. You can obtain information about the sources or medium from where you are acquiring users and traffic to your APIs, your most popular developers applications, and request verbs. By default, it displays data for the last two months and present month until the current date.

At the top of the Usage report page, there is a date-range selector. This date-range selector lets you set the time period for which you want to analyze the reports. To set a new time period, click and drag the bubble-like endpoints available on the date-range selector.

Above the date-range selector, you can select **Month**, **Week**, or **Day** tabs to see data by month, week, or day.

The **Month** option displays three touch points, one for each of the last three months inclusive the current month.

The **Week** option displays one touch point for each week of the last three months inclusive the current month. A week starts on a Sunday and ends on a Saturday.

The **Day** option displays one touch point for each day. The number of touch points displayed here varies depending upon the time range you have selected under Month or Week tabs.

At the top right corner of the Usage page, click to view advanced filter menu and options. The filter menu and options appear below the date-range selector and you can filter your reports by API, Applications, Products, or Developers. Once you apply the filter options, the applied filters are displayed under **Active Filters**.

The rest of the Usage page displays a graphical view of key API metrics, which include the following:

- API Calls
- Developer Engagement Status
- New Developers
- New Applications
- Top Browsers
- Top Agents
- Top Operating Systems
- Top Device Types
- Browser Call Count
- Agent Call Count
- Operating Systems Call Count
- Device Types Call Count
- Request Verb Call Count

**Note**

In any of the key API metrics, if you observe an entity in the chart marked as **Unidentified**, it indicates the following:

- **Unidentified Application**: An application could not be identified for these API calls as the API key could not be verified.
- **Unidentified Product**: A product could not be identified for these API calls as the API key could not be verified.
- **Unidentified Developer**: A developer could not be identified for these API calls as the API key could not be verified.
- **Unidentified Platform**: A platform could not be identified for these API calls as the API key could not be verified.
- **Unidentified Agent**: An agent could not be identified for these API calls as the API key could not be verified.
- **Unidentified Operating System**: An operating system could not be identified for these API calls as the API key could not be verified.
- **Unidentified Device Type**: A device type could not be identified for these API calls as the API key could not be verified.

**Date-Range Selector**

In the **Health** and **Usage** report pages, there is a date-range selector. This date-range selector lets you set the time period for which you want to analyze the reports. To set a new time period, click and drag the bubble-like endpoints available on the date-range selector.

At the top-right corner of the date-range selector, there is a small action bar with options to hide the date-range selector and refresh the reports.

Click 🎧 to hide or unhide the date-range selector.

Click ⚡ to refresh the reports with latest data from API calls.

While viewing your reports on the Health and Usage page, you can choose to keep the data-range selector always visible. You can do so by clicking on the 🔄 icon available below the date-range selector.
**Action Bar**

The Action Bar appears at the top of each report that contains graphical data. The controls on the action bar allow you to act on the graphical data. Using these controls, you can switch between graphical view and tabular view, and switch between different chart types.

- Click to hide and unhide legends for a graph.
- Click to view enlarged and diminished image of a graph.
- Click to switch between full screen view and default screen view.
- Click to select a different chart type. You can select between pie charts, line charts, bar charts, donut charts, or heatmaps. Note that not all chart types might be supported for a specific report.
- Click to switch between a tabular view and graphical view of a report.

### Related Information

- [Creating and Working with Custom Reports](page 498)
- [Creating Custom Dimensions and Measures](page 501)

### 1.5.3.2.2 Creating and Working with Custom Reports

Create your own custom reports in Advanced API Analytics dashboard.

### Context

You can create customized charts for API metrics that are critical to your business. Using the Custom view feature, you can group all these API metrics and view them in a single window.
Procedure

1. In the analytics dashboard, choose Custom View from the +Add dropdown menu.
2. In the Create Custom View dialog, enter a name for your new custom report and choose OK.
3. In the Create Chart window, enter a title and a description for the chart that you want to create.
4. Under Dimensions, choose the API metric that you want to measure from the dropdown menu.
5. Under Measures, choose how you want to measure the selected API metric.

For example, you want to create a chart to plot the number of API calls received through a device type. In this case, you can name the chart as Number of calls per device, and choose Device type under Dimensions, and choose Calls under Metrics.

Adding multiple dimensions and measures

You can add multiple measures with various dimensions for a single chart. For example, you can plot the total number of calls and errors occurring on a particular device type. To do so, you can select device type under Dimension and add two entries under Measures, one for tracking the number of calls and the other for tracking the number of errors.

To view the chart for a particular measure, select the measure from Selected Measures dropdown menu available at the top of the chart. In addition, the dimension and measures, and its details appear as a table below the chart.

You can also add multiple dimensions. For example, if you want to plot the number of calls from a particular device type and from an operating system, then you can add two entries under Dimensions, one for the device type and the other for the Operatings system type.

To drill down on a particular dimension, click the corresponding bar on the plotted chart. If you apply a filter to a chart to drill down to the details, you can navigate back to any previous parameter by using the breadcrumb option.

If you want to analyze using a particular value of measure or dimension in the plotted chart, choose the Filter icon at the top of the chart and set the required values in the Filter popup. You can enter values for measures manually using the Equals to and Not Equals to options. For static dimensions (default), you can choose multiple values from the dropdown list.

6. Choose OK and then save the chart by choosing Save.

The newly created custom view appears as a new report page next to the default report pages. To view all your custom charts and reports, choose the created custom view report page. You can create a maximum of three custom views.

To edit the name of a custom view, select the required custom view and click on the icon displayed next to the custom view name.

7. To add new charts to your custom view, choose the required custom view and click Create.

8. In the Create Chart window, enter a name and a description for the chart in the Title and Description fields, and under Dimensions and Measures, choose the API metric that you want to track and measure.

At the top of your custom report page, there is a date-range selector. This date-range selector lets you set the time period for which you want to analyze the reports. To set a new time period, click and drag the bubble-like endpoints on the date-range selector.

At the top of the date-range selector, select Month, Week, Day, Hour, or Minutes to see data by month, week, day, or hour.
The **Month** option displays six touch points, one for each of the last six months inclusive the current month.

The **Week** option displays one touch point for every week of a month.

The **Day** option displays one touch point for each day.

The **Hour** option displays 24 touch points, one for each hour of the day.

The **Minutes** option displays 48 touch points, one for every 30 minutes of the day.

9. At the top-right corner of the date-range selector, you find options to hide the date-range selector, view a grid representation of all your custom charts, and refresh the reports.

   Click 📬 to hide or unhide the date-range selector.

   Click 🌐 to view a grid representation of all the charts that are a part of your custom view. Hovering over each chart gives you options to either remove the chart from your custom view or add the chart to your custom view.

   Click 🔁 to refresh the reports with latest data from API calls.

   Click 💩 to delete a custom view.

   Deleting a custom view deletes all its associated charts and data.

10. Each custom chart that you add to your custom view provides an action bar with options to edit and delete the chart.

   Click ✒️ to edit the chart.

   Click 🗑️ to delete the chart.

**i Note**

In any of the custom charts, if you choose **Line Chart** or **Stacked Bar Chart** chart types, the custom chart displays a time-wise trend of the report. For example, if you have a custom chart created for displaying the number of calls per API, then selecting the **Line Chart** type displays the number of calls based on the time period (Month, Week, or Day) you have selected.

**Related Information**

- Find Your Way around Advanced API Analytics Dashboard [page 493]
- Creating Custom Dimensions and Measures [page 501]
1.5.3.2.3 Creating Custom Dimensions and Measures

Capture and analyze data using custom dimensions and custom measures.

Context

Advanced API Analytics provides a set of default dimensions and measures to track analytics data. However, if you need dimensions and measures that aren’t included in the default list, you can create custom dimensions and measures.

With a custom dimension or a custom measure, you collect and analyze data that analytics don’t automatically track. For instance, you want to capture API calls or API errors based on an API Key. Advanced API Analytics doesn’t provide an out-of-the-box dimension that allows you to track data based on an API key. In such cases, you can define a custom dimension for capturing API-Key-based data. Similarly, you want to track the number of headers passed in an API call. In such cases, you can create a custom measure to track the total or average number of headers passed in an API call.

Procedure

1. In the analytics dashboard, choose Custom Metric from the +Add dropdown menu.
2. In the Add Custom Metric window, enter the name of the custom dimension or the custom measure that you want to add for tracking data. In this step, you enter just the names of custom dimensions and measures. However, for enabling data collection with them, you must reuse the names of custom dimensions or measures in the Statistics Collector policy of your API proxy. This procedure is explained in further steps.
3. Choose OK.
4. Choose the navigation icon on the left and choose Develop.
5. From the APIs list, choose the required API for which you want to collect data using the custom metric.
7. Attach the Statistics Collector Policy [page 306] to the PreFlow of your ProxyEndpoint. For more information about how to add policies to API proxy, see Policies [page 92].
8. Open the payload of Statistics Collector policy that you attached to the API Proxy.

i Note

By default, the payload of Statistics Collector policy displays all the custom dimensions and measures that you’ve created. It displays them in a commented state with xml indicators <!-- --> as shown in the below sample payload:

Sample Code

<!-- The policy collects data for each request and passes to the analytics server. In the below sample payload, you can see a custom dimension 'APIKey' for
To enable data collection, you must uncomment the custom dimension or the measure with which you want to enable data tracking. In the below sample payload, data collection is enabled only for the custom dimension `APIKey`.

**Sample Code**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<StatisticsCollector xmlns="http://www.sap.com/apimgmt">
  <Statistics>
    <Statistic name="APIKey" ref="request.header.APIKey" type="string">999999</Statistic>
    <!-- <Statistic name="HeadersCount" ref="request.headers.count" type="integer">0</Statistic> -->
  </Statistics>
</StatisticsCollector>
```

9. After you’ve created the custom dimension or measure, navigate to the analytics dashboard. Add a custom view and create custom charts using the custom dimensions or measures you created.

**Note**

After creating a chart with custom dimension or custom measure, you’ll experience a delay of 20-30 minutes before data starts appearing in the charts.

**Related Information**

- Advanced API Analytics [page 492]
- Creating and Working with Custom Reports [page 498]

**1.5.3.3 SAP Analytics Cloud for API Management**

SAP Analytics Cloud is an enterprise-wide solution that combines business intelligence, planning, and predictive analytics into one cloud environment. It provides a unified and secure public cloud experience to the users enabling faster decision making. For more information, see [SAP Analytics Cloud](#).

**API Management Reporting Dashboard**
API Management extends the Integration Suite capability by providing a digital experience to design, develop, and manage APIs in a secure and scalable environment. Therefore, you can use the API Management Reporting dashboard, which is built upon SAP Analytics Cloud to quickly glance through and analyze important API metrics in the form of charts and KPIs. For more information, see SAP Integration Suite.

Related Information

Overview
Architecture and Abstract
Stories - API Management
Models
Data Connectivity

1.5.4 Consume APIs

Consume APIs via the API business hub enterprise. In the API business hub enterprise, an application developer registers, explores the API exposed by customers, creates applications, and tests APIs.

API business hub enterprise is an application that provides a common platform for Application developers to consume APIs. Every API Management customer is provided with their own API business hub enterprise.
application on cloud. The API business hub enterprise offers capabilities to onboard application developers, explore and test APIs, create and subscribe to Applications.

The API business hub enterprise supports the following features:

- **Onboard an Application developer** - To explore the APIs and subscribe to an Application, an Application developer must be registered to the API business hub enterprise. On registering, the Application developer is provided access to the API business hub enterprise.
- **Browse Catalog** - Explore the Products (assembled APIs) available in the Catalog store, navigate to individual API proxies, read the API Documentation, and view the resources attached to the APIs.
- **Create Applications** - An Application developer can create one or more applications to consume APIs. To consume the APIs, an Application developer must subscribe to an Application (assembled Products). It is by subscribing to an Application that you return to the developer the key required to access the APIs.
- **Download SDK** - You can also download the client software development kit (SDK) for developers through a non-commercial license on open source sites. You can use this SDK for developing applications.
- **Test APIs** - You can test the APIs and understand the runtime behavior of the APIs better. Use the Test Console to explore the resources associated with an API and execute the operations.

### 1.5.4.1 Onboard an Application Developer

Explains how API administrators can onboard Application developers so they can access the API business hub enterprise.

#### Context

To provide Application developers with access to the API business hub enterprise, the API Administrator first has to onboard them. The steps to onboard an Application developer are as follows:

#### Procedure

1. The Application developers log on to the API business hub enterprise application with their IDP user credentials, and register to the API business hub enterprise. For more information, see Register on API business hub enterprise [page 505].
2. The API Administrator approves or rejects the request to access the API business hub enterprise. For more information, see Approve or Reject Access Requests [page 506].

   If you haven’t enabled the automatic creation of shadow users, and you’ve not explicitly created shadow users for your developers, then they’re unable to log on to the application, and they’re asked to contact the administrator. For more information, see Creation of Shadow Users [page 72]
1.5.4.1.1 Register on API business hub enterprise

Procedure to register as an Application developer on the API business hub enterprise to view the products available in the Catalog store. The API business hub enterprise also enables you to explore the APIs, read the associated API Documentation, and view resources.

Prerequisites

- **As a Developer you’re trying to self-register:**
  - You’re already a valid Application IDP user.
  - The admin has already added your email ID in the subaccount.

- **As an Admin you’re trying to onboard multiple users:**
  - The admin has already added the email IDs of the users in the subaccount.
  - The admin has assigned the `AuthGroup.API.Admin` role to all the users.

  **Note**

  While onboarding multiple users, it is recommended that you don’t assign the `AuthGroup.API.Admin` role to all the users as this will enable the developers to take on the admin role. Instead you can automate the process of onboarding multiple users by using the API “API Business Hub Enterprise - Registering Users(CF)”. In this case, admin approval is not required. When the user logs in and chooses the Register button, they get auto registered as developers.

  **Note**

  The `AuthGroup.API.ApplicationDeveloper` role must not be assigned manually to a user form the SAP BTP Cockpit and this role must not be a part of any user group assignment. This role is assigned by default to a user who onboards to the API business hub enterprise using the Self-registration process or via Add User flow. See Assigning User Roles [page 64], for more information.

Context

The procedure below describes the sequence of steps when as a developer you’re trying to self-register:

Procedure

1. Log on to the API business hub enterprise application with your IDP user credentials.
2. To register to the API business hub enterprise as an Application developer, choose Register.

   A dialog box with the prepopulated data such as, your first name, last name, and e-mail address appears.
3. Enter the country/region and reason for requesting access to the API business hub enterprise.

4. Choose OK.

The request is sent to the administrator with the AuthGroup.API.Admin role.

- If the administrator approves your request, you’ll receive an e-mail notification. You can log in to the API business hub enterprise via the link provided in the e-mail.
- If the administrator rejects the request, you’ll receive an e-mail notification with the reason for the rejection. When you log on to the application, you’ll see the reason for request rejection on the display page.

**Note**

Application Developers can now email to the administrator by replying to the email notification they receive for any queries regarding their access request to the API business hub enterprise application.

### 1.5.4.1.2 Approve or Reject Access Requests

Procedure to provide or reject access to an Application developers for using the API business hub enterprise.

**Prerequisites**

You are assigned the *AuthGroup.API.Admin* role.

**Context**

As an API administrator, you use this procedure to provide or reject access to an Application developers for using the API business hub enterprise.

**Procedure**

1. Log on to the API business hub enterprise.

2. Choose [Manage > Manage Users](#).

   **Note**

   We recommends to use the [Manage Users](#) page only to onboard Application developers. For assigning roles to other users, use the SAP BTP Cockpit.

3. Configure the e-mail id in the [E-mail Configuration](#) textbox to receive requests and send approvals from this e-mail id.

4. To view the pending requests, navigate to [Pending](#) section.
5. In the pending section, accept or reject the request by choosing the corresponding action item in the *Actions* column.

   On accepting the request, an approval email is sent to the requester. On rejecting a request, you need to provide a reason and an email notification is sent to the requester with the mentioned reason.

6. To view registered users, navigate to *Registered* section.

   In the *Registered* section, you can perform the following:
   - Edit an existing user to add or remove user roles.
   - Register a new user by selecting the add user icon.

**Related Information**

Configure the API business hub enterprise [page 508]
Revoke Access [page 507]

### 1.5.4.1.3 Revoke Access

Revoke the access of an application developer.

**Prerequisites**

You are an API administrator and the role AuthGroup.API.Admin is assigned to your user.

**Context**

As an API administrator, you use this procedure to revoke an application developer's access for using the API business hub enterprise.

**Procedure**

1. Log on to the API business hub enterprise.
2. Choose *Manage* > *Manage Users*.
3. Go to the *Registered* section. From the list of application developers, select the application developer whose access you want to revoke and choose the revoke user action item under the *Actions* column.
4. In the *Revoke* window, provide a reason for revoking the access.
1.5.4.2  Configure the API business hub enterprise

You can configure the API business hub enterprise to personalize it for your organization.

Select Manage from the banner. Modify the following sections to personalize the API business hub enterprise.

The prerequisite varies for each section. Sections are visible to the user based on the role assigned to the user. For example, General section is visible to only users assigned with site admin role. For more information on API business hub enterprise user roles, see.

**General**
Prerequisite: You are assigned the site admin role.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Logo</td>
<td>Upload an image file for your logo, and save the changes.</td>
</tr>
<tr>
<td>Color Scheme</td>
<td>Information about the color scheme used.</td>
</tr>
<tr>
<td></td>
<td>When you upload a logo, the dark color of the navigation area at the top of</td>
</tr>
<tr>
<td></td>
<td>the screens is replaced with white, since many logos require a white</td>
</tr>
<tr>
<td></td>
<td>background.</td>
</tr>
<tr>
<td></td>
<td>Other colors on the Web site are changed to neutral tones to avoid visual</td>
</tr>
<tr>
<td></td>
<td>conflicts with your company’s logo colors.</td>
</tr>
</tbody>
</table>

**Home Page**
Prerequisite: You are assigned the site admin role to view Name and Description and Updates. You are assigned the content admin role to view Navigation Categories.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name and Description</td>
<td>Edit the default name and description for your application.</td>
</tr>
<tr>
<td>Updates</td>
<td>Configure updates to be displayed on the Home page. For more information on how to add, edit, or delete an update, see Manage Updates [page 509].</td>
</tr>
<tr>
<td>Navigation Categories</td>
<td>Configure navigation categories to be displayed on the Home page. For more information on how to add, edit, or delete a category, see Manage Navigation Categories [page 510].</td>
</tr>
</tbody>
</table>

**Manage Users**
Prerequisite: You are assigned the admin role.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-mail Configuration</td>
<td>Provide the administrators e-mail id.</td>
</tr>
<tr>
<td>Pending Requests</td>
<td>Information about pending user requests. You can either accept or reject</td>
</tr>
<tr>
<td></td>
<td>the requests. For more information, see Approve or Reject Access Requests</td>
</tr>
<tr>
<td></td>
<td>[page 506].</td>
</tr>
<tr>
<td>Registered Users</td>
<td>Information about users registered. You can either edit the roles for an</td>
</tr>
<tr>
<td></td>
<td>existing user or register a new user. For more information, see Approve or</td>
</tr>
<tr>
<td></td>
<td>Reject Access Requests [page 506].</td>
</tr>
</tbody>
</table>

**Manage API Portal Connection**

Prerequisite: You’re assigned the `AuthGroup.API.Admin` role.

As an API business hub enterprise administrator, you can approve or reject the connection requests from this page. You can also view the requests that you have previously approved or rejected.

**Reference Links**

Prerequisite: You are assigned the site admin role.

You can add, edit, and customize the links that appear at the bottom of the page here. Links are grouped into sets of three. Click + to add and edit Web site updates and news. In the Add Link dialog window, enter the link title and URL, and choose Save.

### 1.5.4.2.1 Manage Updates

Configure the updates to be displayed in the Updates section on the home page.

**Prerequisites**

You have the site admin role assigned to you.

**Context**

Use the following procedure to configure updates.
Procedure

1. Log on to the API business hub enterprise, and navigate to Manage Home Page Updates Configure Updates.
2. To add an update, click Add Update icon. In the Add Update screen that opens, enter the following details:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Provide a title for the update.</td>
</tr>
<tr>
<td>Description</td>
<td>Provide a description for the update.</td>
</tr>
<tr>
<td>Link</td>
<td>Provide the details of a reference link. Link is an optional field to provide more information on the update.</td>
</tr>
</tbody>
</table>
3. Save the changes.

Newly configured update is visible in the Configured Updates section. In this section, you can perform the following:
- Reorder the updates by using the Move Up and Move Down action icons.
- Edit an update by choosing the edit action icon.
- Delete an update by choosing the delete action icon.

1.5.4.2.2 Manage Navigation Categories

Configure the navigation categories to be displayed on the home page.

Prerequisites

You have the content admin role assigned to you.

Context

Use the following procedure to configure navigation categories.

Procedure

1. Log on to the API business hub enterprise, and navigate to Manage Home Page Navigation Categories.
2. To add a category, choose Add Category icon. In the Add Category screen that opens, enter the following details:
### Subscribe to a Product

You can subscribe to a product and add it to an existing application or create a new application.

#### Context

#### Procedure

1. Log on to the API business hub enterprise.
2. Navigate to the required Product. You can search for the required product.
3. In the product details screen, choose **Subscribe**. You can subscribe to:
   - **Add to Existing Application**: The list of applications appears. Choose the required application.
   - **Create New Application**: Create an application by entering the name and title. The selected Product is added to the application by default.
4. Choose **Save**.

### Create an Application

Create an Application to consume the required APIs.

**Prerequisites**
You either have the AuthGroup.API.ApplicationDeveloper role or AuthGroup.API.Admin role assigned to you. For more information on roles, see Assigning User Roles [page 64].

i Note

The AuthGroup.API.ApplicationDeveloper role must not be assigned manually to a user from the SAP BTP Cockpit and this role must not be a part of any user group assignment.

This role is assigned by default to a user who onboards to the API business hub enterprise using the Self-registration process or via Add User flow.

A user must be onboarded to API business hub enterprise only via Self-registration or Add User flow. For more information on registering in API business hub enterprise, see Register on API business hub enterprise [page 505]. In the Add User flow, the API business hub enterprise admin adds a user who wants to be onboarded to API business hub enterprise. However, the user who is requesting to be onboarded must ensure that the user details provided to the admin matches the user details obtained from the response of <developer portal url>/api/1.0/users.

An application is a discrete representation of the actual developer's application. It provides the developer with an API key to pass-in with every request to the API.

In API Management, similar APIs are bundled together to form products, which are published in the Catalog. An application developer enters necessary details to register to the API business hub enterprise. After successful registration, the Application Developer can explore the required products and APIs to create an application. Once the application has been created successfully, the system generates an Application Key and Application Secret. If APIs in the application you created are protected via Verify API Key policy, then to access those APIs, you must pass the generated Application Key. Whereas, if APIs are protected via OAuth policy, then to access those APIs, you must pass an OAuth token that can be obtained by using the combination of generated Application Key and Application Secret.

Creating an Application with Application Developer Role

1. Log on to the API business hub enterprise.
2. Navigate to the My Workspace page.
   - If you have created applications earlier, they're displayed under the Applications section. For a created application, you can view the total number of calls made in the current month.

   i Note
   
   By default, the Cost section displays the cost incurred in the last 6 months and the cost incurred in the current month. However, you can choose a month to view the cost incurred for that month.
   
   You can choose to obtain the latest metering data.

   i Note
   
   You might experience some delay before you see the latest metering data.

   Notation used to display the data is as per metric specifications, for example:
○ 999 shows as 999 and 1000 shows as 1k
○ 999000 shows as 999K and 1000000 shows as 1M
○ 1500000 shows as 1.5M and 1000000000 shows as 1G

3. To create an application, choose + in the Applications section.
4. In the Create an Application dialog, enter a Title, a Description (optional), and a Callback URL (optional) for the application.
5. Choose + to add products to this application.
6. In the Add Products dialog, select the products that you want to associate with the application.
7. Choose OK.
8. Choose Save.

i Note
While creating an application, if you’ve selected the Take me to this new application now checkbox, you’re directly navigated to the newly created application.

The application you have created appears under the Applications section, and also under the Applications tab in API portal.

i Note
If you open any created application, you notice that the system has generated an API Key automatically. Use this value to access the API. At any point in time, you can regenerate the API Key using Regenerate Key option. When you regenerate the key, both Application key and Secret key are changed. When you trigger API using the old key, then the response is negative. The old API key becomes invalid on regeneration.

Creating an Application with API business hub enterprise Administrator Role

A API business hub enterprise Administrator can:

• create an application on behalf of a user (Application Developer) and handover the application key and secret to that user.
• create new applications in different landscapes(example: production, nonproduction) by maintaining the same application key and secret.
• create custom attributes at application level and regulate the API call logic

1. Log on to the API business hub enterprise.
2. Navigate to the My Workspace page.
If you or other application developers have created applications earlier, they’re displayed under the Applications section. For a created application, you can view the total number of calls made in the current month.

**i Note**

By default, the Cost section displays the cost incurred in the last 6 months and the cost incurred in the current month. However, you can choose a month to view the cost incurred for that month.

**i Note**

For API business hub enterprise administrators, analytics data is unavailable for those applications that they created on behalf of other users or application developers.

You can choose 🔄 to obtain the latest metering data.
Note
You might experience some delay before you see the latest metering data.

Notion used to display the data is as per metric specifications, for example:
- 999 shows as 999 and 1000 shows as 1k
- 999000 shows as 999K and 1000000 shows as 1M
- 1500000 shows as 1.5M and 1000000000 shows as 1G

3. To create an application, under Applications section, choose + .
4. In the Create an Application dialog, enter a Title, a Description (optional), and a Callback URL (optional) for the application.
5. As an administrator, you have the option to create an application on behalf of a user (Application Developer). To achieve this task, select the Create this application on behalf of someone else checkbox and enter the User ID of the user on behalf of whom you are creating the application. If you already possess an application key and secret, then select the Already have Application Key and Secret checkbox and enter the Application Key and Application Secret .

Note
Application key and secret of an existing org can't be used in a new application in a new org. This implies that you'll not be able to use the same application key and secret in multiple orgs within the same data center and region.

6. Choose + to add products to this application.
7. In the Add Products dialog, select the products that you want to associate with the application.

Note
You can select multiple products.

8. Choose OK.
9. Choose Save.

Note
While creating an application, if you’ve selected the Take me to this new application now checkbox, you’re directly navigated to the newly created application.

The application you created appears under the Applications section, and also under the Applications tab in API portal.

Note
If you open any created application, you notice that the system has generated an API Key automatically. Use this value to access the API. At any point in time, you can regenerate the API Key using Regenerate Key option. When you regenerate the key, both Application key and Secret key are changed. When you trigger API using the old key, then the response is negative. The old API key becomes invalid on regeneration.
10. Specify custom attributes.
    1. Under My Workspace, choose an application for which you want to add custom attributes.
    2. In the Application Info screen, under Custom Attributes section, choose to add a custom attribute.
    3. In the Add Custom Attribute dialog, enter a name and a value for your custom attribute and choose Add.

**Note**
You can create a maximum of 18 custom attributes per application. You cannot modify the name of a created custom attribute. However, you can modify its value whenever required. You can delete a custom attribute if it is no longer needed.

For more information on the usage of custom attributes in an application, see Example: Accessing the Custom Attributes of an Application [page 516].

### 1.5.4.4.1 Example: Accessing the Custom Attributes of an Application

Let’s say as a Developer Portal Administrator, you would want to restrict the number of calls to an application based on Application Key. To achieve this result, you create two applications Application_1 and Application_2.

- **Application_1** contains two products namely Prod_1 and Prod_2.
- **Application_2** contains two products namely Prod_3 and Prod_4.

Prod_1 and Prod_2 contain two common APIs namely API_1 and API_2.

For **Application_1**, add the following custom attributes and its corresponding values:

- app_time_unit = minute
- app_quota_interval = 1
- app_quota_count = 9

For **Application_2**, add the following custom attributes and its corresponding values:

- app_time_unit = minute
- app_quota_interval = 1
- app_quota_count = 5

To leverage these custom attributes in your API proxy execution, you must:

- add a verify API Key policy to the APIs that are part of your application.
- add a Quota policy to APIs that are part of your application.

For **API_1** and **API_2**, add the following sample policy payloads:

Sample payload for Verify API Key policy:
Sample payload for Quota policy:

```xml
<!-- can be used to configure the number of request messages that an app is allowed to submit to an API over a course of unit time -->
<Quota async="false" continueOnError="false" enabled="true" type="calendar" xmlns="http://www.sap.com/apimgmt">
    <Identifier ref="verifyapikey.Verify-api-key.developer.id"/>
    <!-- specifies the number of requests allowed for the API Proxy -->
    <Allow countRef="verifyapikey.Verify-api-key.app_quota_count"/>
    <!-- the interval of time for which the quota should be applied -->
    <Interval ref="verifyapikey.Verify-api-key.app_quota_interval"/>
    <!-- used to specify if a central counter should be maintained and continuously synchronized across all message processors -->
    <Distributed>true</Distributed>
    <!-- Use to specify the date and time when the quota counter will begin counting, regardless of whether any requests have been received from any apps -->
    <StartTime>2015-2-11 12:00:00</StartTime>
    <!-- if set to true, the distributed quota counter is updated synchronously. This means that the update to the counter will be made at the same time the API call is quota-checked -->
    <Synchronous>true</Synchronous>
    <!-- Use to specify the unit of time applicable to the quota. Can be second, minute, hour, day, or month -->
    <TimeUnit ref="verifyapikey.Verify-api-key.app_time_unit"/>
</Quota>
```

**Note**

The attribute names `app_quota_interval`, `app_quota_count`, and `app_time_unit` must be the same attributes that you have added while creating the application.

To verify if the custom attributes are used in runtime, make an API call with `<appKey_1>` passed as a query parameter. For example, `https://<API_proxy_URL>?apikey=<appKey_1>`.

Call the same URL repeatedly and after 9 successive calls, your API proxy must return a Quota violation message.

Similarly, make an API call with `<appKey_2>` passed as a query parameter. For example, `https://<API_proxy_URL>?apikey=<appKey_2>`.

Call the same URL repeatedly and after 6 successive calls, your API proxy must return a Quota violation message.
1.5.4.5 Consume Applications

Once you create an Application, you can then consume the APIs based on your business requirements.

On subscribing to an application, the application developer receives an API key that the application must pass on every request to the API. API keys provide a simple mechanism for authenticating applications.

API Management generates API keys for applications, and enables you to add API key-based authentication to your APIs using policies. However, enforcement of the key is performed at the API proxy level, not by the API product itself. Therefore, you must ensure that all API proxies, and the corresponding resources defined by those API proxies, implement some form of key validation.

Before you use the API keys, ensure you are aware of the policies that support API keys and their functionality. There are two popular ways how APIKeys are provisioned. They are provided either as part of a Simple APIKey verification or as part of OAuth verification.

VerifyAPIKey Validation

If you define an API proxy to perform key validation by using the VerifyAPIKey policy, provide the API Key details to gain access to the applications.

OAuth 2.0 Validation

API Management supports standard OAuth flow. Currently SAP supports only Client credentials grant_type in OAuth.

Before you start, make a note of the Application key and secret for the required application.

A request is made using the following:

- URL: <URL of OAuth token>
- Method: POST
- Custom Header: Authorization value: Basic <Application key>:<Application secret>(base64 encoded)
- Payload: grant_type=client_credentials

This call returns a json payload with the OAuth validation response. On successful validation, it contains the access token. Note down the access token.

As default, the expiry time is configured to 3600 secs (1 hr). You can also configure the expiration time and the details that have to be displayed as part of the response.

You can now use this access token to fetch OAuth enabled services while making the actual business API call:

- URL: http[s]://<host>:<port>/<service_path>
- Custom Header: Authorization value: Bearer <access_token>

If the access token is valid, a valid service response is returned.
1.5.4.6 Analyze Applications

Use analytics capabilities to analyze application usage, performance, and error count.

API Management provides comprehensive analytics capabilities to understand application consumption. The runtime data is gathered, analyzed, and displayed as charts, headers, and key performance indicators (KPIs).

As an application developer, navigate to My Workspace to view the analytics information. By default, the analytics section displays the data for all the applications subscribed by you. All charts are displayed based on the Application Developer context.

The analytics information can be viewed as Performance Analytics and Error Analytics.

- **Performance Analytics**: Displays the performance-related charts and KPIs for the selected time period. Following table describes the charts used to analyze the performance of all applications:

<table>
<thead>
<tr>
<th>Chart Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic Across all APIs</td>
<td>This chart displays total API calls made across all applications.</td>
</tr>
<tr>
<td>Slowest APIs</td>
<td>This chart displays the slowest APIs based on the API response time.</td>
</tr>
<tr>
<td>Top APIs</td>
<td>This chart displays most frequently used APIs.</td>
</tr>
<tr>
<td>Top Products</td>
<td>This chart displays most frequently used products based on the number of calls made to the APIs associated with the product.</td>
</tr>
<tr>
<td>Top Applications</td>
<td>This chart displays most frequently used applications based on the number of calls made to the APIs associated with the application.</td>
</tr>
</tbody>
</table>

- **Error Analytics**: Displays the error-related charts and KPIs for the selected time period. Following table describes the charts used to view error analytics of all the applications:

<table>
<thead>
<tr>
<th>Chart Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Errors</td>
<td>This chart displays total errors.</td>
</tr>
<tr>
<td>Error Prone APIs</td>
<td>This chart displays number of errors per API.</td>
</tr>
<tr>
<td>Error Prone Applications</td>
<td>This chart displays number of errors per API associated with the application.</td>
</tr>
</tbody>
</table>

To view analytics for a specific application, navigate to the application details screen by selecting the required application. However, in the application details screen the analytics information is available only for the following KPIs:

- Traffic Across all APIs
- Slowest APIs
- Error Prone APIs
1.5.5 Monetize APIs

API Management provides monetization feature to all API providers to generate revenue for using the APIs.

As an API Admin, you can create a rate plan, attach it to a product in the API portal, and publish the product in the API business hub enterprise. You can also view bill details of each developer in the API portal.

As an application developer, in the API business hub enterprise you can create an application and add products to the application. Based on the product usage, you can view the corresponding bill details.

API Management provides this feature through the following services:

- Rate Plan Service [page 520]
- Billing Service [page 525]

If you were creating, updating, or reading an application using the APIs and not through the user interface, then you need to switch to Subscription entity from Application entity to use the Monetize feature. For more information see, Create or Update or Read an Application using Subscription key [page 529]

1.5.5.1 Rate Plan Service

API Management allows user to create rate plans and attach a rate plan to a product. Through a rate plan you can charge the application developers for the use of your APIs.

For more information see,

- Create a Rate Plan [page 520]
- Attach Rate Plan to a Product [page 522]
- Update a Rate Plan [page 523]
- Delete a Rate Plan [page 525]

1.5.5.1.1 Create a Rate Plan

Create a rate plan using the API portal.

Prerequisites

You are assigned the admin role.

Context

You are creating a rate plan.
Procedure

1. Log on to the API portal.
2. From the navigation bar, choose Monetize.
3. On the Monetize screen, choose Create.
4. On the Create Rate Plan screen, enter values for the following fields:
   - **Name**: Name of the rate plan.
   - **Description**: Outline of the plan.
   - **Frequency**: Monthly
   - **Currency**: Euro
   - **Basic Charge**: Minimum bill amount paid by the user after subscribing to the product associated with this rate plan.
   - **Rate per API Call**: Amount in Euros for one API call.
   - **Plan Type**: Choose either Basic or Tier.
     - **Basic**: In Basic rate plan type, the rate charged per API call is fixed.
     - **Tier**: In Tier based rate plan type, the rate charged per API call varies based on the number of API calls.

   **Example**
   Tier based rate plan

<table>
<thead>
<tr>
<th>API Calls From</th>
<th>API Calls To</th>
<th>Rate per API Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5000</td>
<td>0.0</td>
</tr>
<tr>
<td>5001</td>
<td>10000</td>
<td>0.5</td>
</tr>
<tr>
<td>10001</td>
<td></td>
<td>0.7</td>
</tr>
</tbody>
</table>

   In the above example, for initial 5000 calls the rate charged is 0.0 per API call. For the next 5000 calls the rate charged is 0.5 per API call and for 10000 + calls the rate charged is 0.7 per API call. For instance, if 8000 calls are made, then the rates per API call is 0.0 for 0-5000 calls and 0.5 for the remaining 3000 calls.

   **i Note**
   If the API Calls To field is left empty, then the system considers the field value to be unlimited.

5. Choose Save.

Related Information

Attach Rate Plan to a Product [page 522]
1.5.5.1.2  Attach Rate Plan to a Product

Attach a rate plan to a product using the API portal.

Prerequisites

- You are assigned the admin role.
- You have created a rate plan in the API portal.

**Note**

You can only attach rate plans to those products that do not have any rate plans associated with them. A product can only be associated with one rate plan. You can also attach a rate plan to a product during the product creation.

Context

You are attaching a rate plan to a product.

Procedure

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Develop.
3. On the Develop page, choose Products.
4. From the list of products available, select the product to which you want to add the rate plan.
5. On the Product details screen, choose RATE PLAN.
7. In the Add Rate Plan window, select the required rate plan from the list of available rate plans.
   You can click an individual rate plan to view the description and details of that particular rate plan.
8. Choose OK.

  Rate plan will not be applicable to existing applications associated with the product to which the rate plan is attached. However, the rate plan will be applicable, if a new application uses the product to which the rate plan is attached.
1.5.5.1.3 Update a Rate Plan

Update a rate plan using the API portal.

Prerequisites

You are assigned the admin role.

Context

You are updating a rate plan.

Procedure

1. Log on to the API portal.
2. From the navigation bar, choose Monetize.
3. On the Monetize screen, choose RATE PLANS.
4. On the RATE PLANS screen, select the rate plan that you want to update.
5. Choose Edit. You can update the following fields only:
   ○ Name: Name of the rate plan.
   ○ Description: Outline of the plan.
   ○ Frequency: Monthly
   ○ Currency: Euro
   ○ Basic Charge: Minimum bill amount paid by the user after subscribing to the product associated with this rate plan.
   ○ Rate per API Call: Amount in euros for one API call.
   ○ Plan Type: Choose either Basic or Tier.
      ○ Basic: In Basic rate plan type, the rate charged per API call is fixed.
Tier: In Tier based rate plan type, the rate charged per API call varies based on the number of API calls.

**Example**

Tier based rate plan

<table>
<thead>
<tr>
<th>API Calls From</th>
<th>API Calls To</th>
<th>Rate per API Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5000</td>
<td>0.0</td>
</tr>
<tr>
<td>5001</td>
<td>10000</td>
<td>0.5</td>
</tr>
<tr>
<td>10001</td>
<td></td>
<td>0.7</td>
</tr>
</tbody>
</table>

In the above example, for initial 5000 calls the rate charged is 0.0 per API call. For the next 5000 calls the rate charged is 0.5 per API call and for 10000 + calls the rate charged is 0.7 per API call. For instance, if 8000 calls are made, then the rates per API call is 0.0 for 0-5000 calls and 0.5 for the remaining 3000 calls.

**Note**

If the **API Call To** field is left empty, then the system considers the field value to be unlimited.

6. Choose Save.

**Note**

Updated rate plan is applicable for new subscriptions only.

**Related Information**

- Create a Rate Plan [page 520]
- Update a Rate Plan [page 523]
- Delete a Rate Plan [page 525]
1.5.5.1.4 Delete a Rate Plan

Delete a rate plan using the API portal.

Prerequisites

You are assigned the admin role.

Context

You are deleting a rate plan.

Procedure

1. Log on to the API portal.
2. From the navigation bar, choose Monetize.
3. On the Monetize screen, choose RATE PLANS.
4. In the Actions column, choose  to delete a rate plan.
5. Choose Yes.

Note

Deleted rate plan is not available for new subscriptions, but is available for existing subscriptions.

Related Information

Create a Rate Plan [page 520]
Update a Rate Plan [page 523]
Attach Rate Plan to a Product [page 522]

1.5.5.2 Billing Service

Billing service is available in both API portal and API business hub enterprise.

Using billing service, you can view the bill details and download bill details for a specific developer and for a specific month.
Service to view bills:

- URL: https://<consumer API-Portal host>:<port>/api/1.0/apimgmt/monetize/bills
- The following table describes the query parameters required to view the bill details.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Required in API portal</th>
<th>Required in API business hub enterprise</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
<td>Yes</td>
<td>Yes</td>
<td>Month in MM format</td>
<td>Month = 03</td>
</tr>
<tr>
<td>Year</td>
<td>Yes</td>
<td>Yes</td>
<td>Year in YYYY format</td>
<td>Year = 2017</td>
</tr>
<tr>
<td>developer_id</td>
<td>Yes</td>
<td>No</td>
<td>Developer e-mail Id</td>
<td>developer_id = <a href="mailto:jon.doe@sap.com">jon.doe@sap.com</a></td>
</tr>
<tr>
<td>application_id</td>
<td>No</td>
<td>No</td>
<td>Id of a specific application for which bill has to be generated</td>
<td>application_id = 6C7F88BB-74BE-4CC-C-A49A-6A8F2BF1EAC1</td>
</tr>
</tbody>
</table>

- You can also view the bill details in the API portal and API business hub enterprise. For more information see:
  - View Bill Details in the API portal [page 526]
  - View Bill Details in the API business hub enterprise [page 527]

### 1.5.5.2.1 View Bill Details in the API portal

View bill details in the API portal for all the applications and products assigned to a particular developer.

**Prerequisites**

You are assigned the admin role.

**Procedure**

1. Log on to API portal.
2. From the navigation bar, choose Monetize.
3. On the Monetize screen, choose BILLS.
4. Select the billing month from the Month and Year dropdown boxes.
   - By default, the bill details for the current month are displayed.
From the list of developers, select the developer you want to view the bill details for.

The Bill Detail window shows a list of applications the developer is subscribed to and the corresponding bill amount for each application. You can view the list of products assigned to each application by clicking the application.

1.5.5.2.2 View Bill Details in the API business hub enterprise

View the bill details in the API business hub enterprise for all the applications subscribed by a developer.

**Prerequisites**

You are assigned the application developer role.

**Procedure**

1. Log on to the API business hub enterprise.
2. Navigate to, My Workspace.
3. The Cost section displays the billing details for all the application subscribed by the developer in two charts:
   ○ Aggregated Costs in Euros: displays the bill details for the last six months.
   ○ Cost for selected month: displays the cost for the selected month. By default, the cost for the current month is displayed.

   To view the cost for a specified application. Navigate to the required application details screen, by choosing the required application. Cost pertaining to that application is visible in the Cost section in the detailed application screen.

1.5.5.2.3 Download Bill Details from API portal

Download bill details using the API portal.

**Prerequisites**

You are assigned the admin role.
**Context**

To download the bill details, proceed as follows:

**Procedure**

1. Log on to API portal.
2. From the navigation bar, choose *Monetize*.
3. On the *Monetize* screen, choose *BILLS*.
4. Select the billing month from the *Month* and *Year* dropdown boxes.
   By default, the bill details for the current month is displayed.
   
   ![Note](image)
   
   You cannot download the bill details for the current month.
   
5. In the Actions column, choose icon to download the bill details.
   Alternatively, you can download the bill details by choosing the *Total Bill Amount* for a developer. Choose *Download* in the *Bill Detail* window.
   
   The bill details are downloaded in a .csv file format.
   
   In the .csv file generated, leading = signs that might have been present in any user input (such Product Title, Application Title, or Rateplan Title) are trimmed.

**1.5.5.2.4 Download Bill Details from API business hub enterprise**

Download bill details using the API business hub enterprise.

**Prerequisites**

You are assigned the application developer role.

**Context**

To download the bill details, proceed as follows:
**Procedure**

1. Log on to the API business hub enterprise.
3. In the *Cost* section, choose the billing month from the *Aggregated Costs in Euros* chart. By default, the cost for the current month is displayed.

   **i Note**
   
   You cannot download the bill details for the current month.

4. Choose *Download* action item.

   The bill details are downloaded in a .csv file format.
   
   In the .csv file generated, leading = signs that might have been present in any user input (such as Product Title, Application Title, or Rateplan Title) are trimmed.

1.5.5.3 **Create or Update or Read an Application using Subscription key**

Creating, updating, and reading an application using the Subscription key.

You can use the following metadata for Subscription entity.

**i Note**

To use the monetization feature, it is recommended to use Subscription entity and not the Application entity to create or update or read an application.

**Sample Code**

```xml
<EntityType Name="SubscriptionsType">
  <Key>
    <PropertyRef Name="id"/>
  </Key>
  <Property Name="id" Type="Edm.String" Nullable="false" MaxLength="256"/>
  <Property Name="reg_id" Type="Edm.String" MaxLength="256"/>
  <Property Name="app_id" Type="Edm.String" MaxLength="256"/>
  <Property Name="product_id" Type="Edm.String" MaxLength="256"/>
  <Property Name="developer_id" Type="Edm.String" MaxLength="256"/>
  <Property Name="ratePlan_id" Type="Edm.String" MaxLength="256"/>
  <Property Name="validFrom" Type="Edm.DateTime"/>
  <Property Name="validTo" Type="Edm.DateTime"/>
  <Property Name="app_name" Type="Edm.String" MaxLength="255"/>
  <Property Name="isSubscribed" Type="Edm.Boolean"/>
  <Property Name="status" Type="Edm.String" MaxLength="255"/>
  <Property Name="comment" Type="Edm.String" MaxLength="2048"/>
  <Property Name="created_by" Type="Edm.String" MaxLength="2048"/>
  <Property Name="modified_by" Type="Edm.String" MaxLength="255"/>
  <Property Name="modified_at" Type="Edm.DateTime"/>
  <NavigationProperty Name="ToApplication"
    Relationship="developer.Subscriptions_ApplicationsType"
    FromRole="SubscriptionsDependent" ToRole="ApplicationsDependent"/>
</EntityType>
```
URL of Subscription Entity: <developer portal base url>/odata/1.0/data.svc/APIMgmt.Subscriptions

Use the below mentioned payload to create an application using Subscription entity:

URL: <developer portal base url>/odata/1.0/data.svc//APIMgmt.Applications

Request Method: POST

Content-Type: application/json

```
{  
  "id": "00000000000000000000000000000000",
  "version": "1",
  "title": "App_Title",
  "description": "Description",
  "callbackurl": "http://www.callbackurl.com",
  "ToSubscriptions": [
    
    {  
      "id": "00000000000000000000000000000000",
      "ToAPIProduct": [
        
        {  
          "__metadata": {
            "uri": "APIMgmt.APIProducts('Product_Catalog')"
          }
        }
      ],
      "ToRatePlan": [
        
        {  
          "__metadata": {
            "uri": "APIMgmt.RatePlans('E8BF82AA-F7B0-427F-881A-D246A047BB00')"
          }
        }
      ]
    }
  ]
}
```

Use the below mentioned payload to update fields like title or callback url or description in the application using Subscription entity:

URL: <developer portal base url>/odata/1.0/data.svc/$batch

Method: POST

Content-Type: multipart/mixed; boundary=batch_349d851f-79ed-44bc-b67a-3159f7cfc17

Content-Length: <length of the content>
Use the below mentioned payload to update the application to add and remove a product using Subscription entity:

**URL:** `<developer portal base url>/odata/1.0/data.svc/$batch`

**Method:** POST

**Content-Type:** multipart/mixed; boundary=batch_ce8f-d810-b289

**Payload:**
```plaintext
--batch_ce8f-d810-b289
Content-Type: multipart/mixed; boundary=changeset_0750-94e8-367a
--changeset_0750-94e8-367a
Content-Type: application/http
Content-Transfer-Encoding: binary

PUT APIMgmt.Applications('238D7CA1-3F61-470B-BC73-37FF311739E2') HTTP/1.1
RequestId: cf1da741-bd68-401e-95d7-9bcf0475112b
Accept-Language: en-US
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
x-csrf-token: CF4601D494334B00239A025C270BDBF1
Content-Type: application/json
Content-Length: 355
--changeset_0750-94e8-367a--
```

**Use the below mentioned payload to update the application to add and remove a product using Subscription entity:**

**URL:** `<developer portal base url>/odata/1.0/data.svc/$batch`

**Method:** POST

**Content-Type:** multipart/mixed; boundary=batch_b72a-e938-270d

**Payload:**
```plaintext
--batch_b72a-e938-270d
Content-Type: multipart/mixed; boundary=changeset_319c-d23e-258e
--changeset_319c-d23e-258e
Content-Type: application/http
Content-Transfer-Encoding: binary

PUT APIMgmt.Applications('238D7CA1-3F61-470B-BC73-37FF311739E2') HTTP/1.1
Accept-Language: en-US
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
Content-Type: application/json
Content-Length: 355
{"id":"238D7CA1-3F61-470B-BC73-37FF311739E2","title":"App_Title_Updated","callbackurl":"http://www.callbackurl_updated.com","description":"Description Updated","app_key":"FuoGpWnZR0SJJedimgRpACH6XaiJc1XR","reg_id":"5f1007f1cd5c47ea8a209ce0567e98f8","version":"1","app_secret":"rp2A0yTgwFGLyeh","valid_from":null,"valid_to":null,"developer_id":"T305297"}
--changeset_319c-d23e-258e--
--batch_b72a-e938-270d--
```
Use the below mentioned url and method to delete an application:

**URL:** <developer portal base url>/odata/1.0/data.svc/APIMgmt.Applications('<app_id>')

**Method:** DELETE

Use the below mentioned url and method to read all applications using Subscription entity:

**URL:** <developer portal base url>/odata/1.0/data.svc/APIMgmt.Applications

**Method:** GET

**Response:** It will fetch only application attributes like title, call back url, description.
- It will not fetch app key and secret.
- If the application is created using Subscription Entity, then the attached products will not be shown here.
- Although, if an application is created using older APIs (only using Application Entity and Application to Product linkage), then the attached product details will be shown in the navigation “ToAPIProductsDetails”.

Use the below mentioned url and method to read a specific applications using Subscription entity:

**URL:** <developer portal base url>/odata/1.0/data.svc/APIMgmt.Applications('<app_id>')?
$expand=ToAPIProductsDetails,ToSubscriptions/ToAPIProduct,ToSubscriptions/ToRatePlan&$format=json

**Method:** GET

**Response:** This API will fetch all the application related details.
• If the application is created using Subscription entity, the associated products will be found in the navigation “ToSubscriptions/ToAPIProduct”.
• If the application is created using older API (only using Application Entity and Application to Product linkage), the attached product details will be found in the navigation property “ToAPIProductsDetails”.

**i Note**

• If a user is not willing to use the Monetization feature, then he or she may continue using Application’s service.
• If a user is willing to use Monetization feature, then he or she must switch to Subscription entity.
• In order to use Monetization, the user needs to create a new rate plan and a new product. Attach the rate plan to the product, publish the product, and let the application developer consume the product via a new application created using Subscription entity.
• It is not recommended to use mix and match of Application entity services and Subscription Entity.
• It is not recommended to use the application created using Application Entity’s service for Monetization purpose.
• It is not recommended to use the application created using Application Entity Service to add a product having rate plan.
• If a user has applications created using Application’s entity service and applications created using Subscriptions entity, then user must make two calls while reading the applications:
  ○ One read call on Application’s entity.
  ○ One read call on Subscription’s entity.

### 1.5.6 Discover Packages

You can discover the API packages supported by the SAP API Management platform that are available in SAP API Business Hub on the API portal.

**Discover**

When you log on to the API portal, you can view the Discover page by choosing the icon. On the Discover page, you can explore packages under the following sections:

• **Highlights**: Showcased packages are displayed in the **Featured** section. Recently published packages are displayed in the **Latest** section.
• **All**: This section lists all the packages in the portal.

Each package is displayed as an individual tile, along with the name of the package, the rating of the package, and a brief description of the package.

If the content has been developed by a partner, the package also displays a **Partner** label.

To find out more about the features of the packages on the API portal, see [Package Details](#page-534).
1.5.6.1 Package Details

A package is a container that can hold different types of content. It typically contains APIs and policy templates. It can also contain documentation and links.

Each package has the following details:

- **Overview**
  The Overview provides the following information about the package:
  - **Description**: Details of the package and the scenarios it can be used for.
  - **Supported Platform**: SAP API Management
  - **Category**: APIs
  - **Created on**: The date and time the package was created.

- **Artifacts**
  The Artifacts section displays the APIs and policy templates available in the package.
  You can view the details of an API, a policy template, or an API proxy by choosing the respective artifact.
  This opens a screen that provides the details of that particular artifact.
  To copy APIs, choose the ![Copy icon] and choose Copy. In the Copy API window, provide the necessary information. You can either leave the information displayed for API Details as it is, or you can change it.

  After the action is completed, you can view the copied API by navigating to Develop → APIS. Alternatively, you can copy the APIs by choosing the respective artifact and then, on the screen that opens, choosing Copy.

  To copy a policy template, choose the ![Copy icon] and select Copy. After the action is completed, you can view the copied policy template by navigating to Develop → POLICY TEMPLATES. Alternatively, you can copy the policy template by choosing the respective artifact and then, on the screen that opens, choosing Copy.

- **Documents**
  This section contains any documents associated with the package.

- **Tags**
  Country, Product, Keyword, Lines of Business, Industry, and other tags for the package are displayed here.

- **Ratings**
  This section contains user ratings and feedback for the package.

- **View in API Business Hub**
  At the package level, you can view and copy APIs to the API portal. If you want to perform actions such as trying out the API or generating the code, then navigate to SAP API Business Hub by choosing View in API Business Hub. Choosing the link takes you to the same package in SAP API Business Hub.

1.5.7 Transport APIs and Its Related Artifacts

API artifacts and their respective application-specific content, can be reused across multiple tenants using the transport mechanism.

You can use the SAP Cloud Transport Management service (TMS) for exporting, importing, and shipping the APIs and its related artifacts from the development or test environment to production environment. For
example, you can design and test API portal content on the test tenant and then use the Cloud Transport Management service to move the content to the target tenant.

This block diagram shows how the content is selected and transported when the Transport Management service is enabled for the first time to transport APIs and its related artifacts from the development or test environment to the production environment:

- Creating Content Assembly Service Destination [page 540]
- Creating API Management Destination [page 543]
- Creating Transport Management Destination [page 548]
- Create a Deploy Service Destination in Cloud Transport Management Service Subaccount [page 550]

**Note**

In this case, you have to create a separate subaccount for Transport Management service.

This block diagram shows how the content is selected and transported when the Transport Management service is already enabled for other SAP offerings and you’re trying to use it for transporting APIs and its related artifacts from the development or test environment to the production environment:
**Note**

In this case, you don’t have to create a separate subaccount for Transport Management service. You can continue to use the Transport Management service in the source subaccount.

Once the user initiates transport of the desired API content from the API portal of the source subaccount, the following events take place:

D1- API Management makes an API call to the Content Assembly Service to inform about the transport.

D2 - Content Assembly Service then makes an API call to fetch the API content from API portal workspace. The Content Assembly Service wraps the API content for transport.

D3- The Content Assembly Service makes a second API call to push the API content to the Transport Management service (TMS).

D4- The SAP Transport Management service makes an API call to the Deploy Service. The Deploy Service calls the API Management in the destination subaccount to import the package into the API Management workspace.
1.5.7.1 Enabling Content Transport Using SAP Cloud Transport Management Service

Configure the service instances and destinations, and establish a route between the source and destination nodes to enable transportation of APIs and its related artifacts.

Prerequisites

- Create a source subaccount and subscribe to API portal, API Management. For more information, see Setting Up API Portal Application [page 43].
- Create a transport subaccount and subscribe to SAP Cloud Transport Management service. Set up and subscribe to SAP Cloud Transport Management service as described in Set Up the Environment to Transport Content Archives directly in an Application. To view and access the SAP Cloud Transport Management service, assign TMS_ADMIN and TMS_VIEWER roles to yourself. To set the roles, scroll down to “Steps to Assign User Roles and Permissions” section in Set Up the Environment to Transport Content Archives directly in an Application.
- Create a Destination subaccount and subscribe to API portal, API Management. For more information, see Setting Up API Portal Application [page 43].

Context

Let us consider a scenario where you want to transport the API Management content from your source subaccount (which is your Development instance) to your destination subaccount (which is your test instance). You must configure the following in the source and the Transport Management subaccounts as shown in the block diagram:

Source subaccount
- An instance of Content Agent
- An instance of API Management, API Portal
- ContentAssembly Service destination
- API Management, API Portal destination
- Transport Management destination

Transport Management service subaccount
- An instance of Transport Management service
- Deploy Service destination

In Note

In case the Transport Management service is already enabled for other SAP offerings, and you’re trying to use it for transporting APIs and its related artifacts, then you don’t have to create a separate subaccount for the Transport Management service. You can continue to use the Transport Management service in your source subaccount.
Further, if you want to transport APIs and its related artifacts from the test to your production instance, your test instance will now become your source. Therefore, you must make all the necessary configurations that you previously made in your development instance in your test instance.

- Creating an Instance of Content Agent [page 540]
- Creating Content Assembly Service Destination [page 540]
- Creating an Instance of API portal, API Management [page 542]
- Creating API Management Destination [page 543]
- Creating Transport Management Destination [page 548]
- Creating an Instance of SAP Cloud Transport Management Service [page 545]
- Create a Deploy Service Destination in Cloud Transport Management Service Subaccount [page 550]

Enabling Content Transport in Cloud Foundry involves the following steps:

<table>
<thead>
<tr>
<th>Steps</th>
<th>Action</th>
<th>Videos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a service instance and a service key of content agent in your source subaccount. See, Creating an Instance of Content Agent [page 540]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Create destination ContentAssemblyService in source subaccount to make API calls to the Content Assembly Service. See, Creating Content Assembly Service Destination [page 540]</td>
<td></td>
</tr>
<tr>
<td>Steps</td>
<td>Action</td>
<td>Videos</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>3</td>
<td>Create an instance of API portal, API Management in your source subaccount and create a service key. See, Creating an Instance of API portal, API Management [page 542]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Create destination APIManagement in your source subaccount to make API calls for fetching the API content from the API portal workspace. See, Creating API Management Destination [page 543]</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Create a service instance and a service key of Transport Management service in your transport subaccount. See, Creating an Instance of SAP Cloud Transport Management Service [page 545]</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Add a Source node in Transport Management Applications. See, Adding a Source Node in Transport Management Applications [page 547]</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Create destination TransportManagement in your source subaccount to make API calls to the Transport Management service (TMS). See, Creating Transport Management Destination [page 548]</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Create a destination in Transport subaccount for the deploy service. See, Create a Deploy Service Destination in Cloud Transport Management Service Subaccount [page 550]</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Add a Destination node in Transport Management Applications. See, Adding a Destination Node in Cloud Transport Management Applications [page 552]</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Create a transport route to connect the source tenant to the destination tenant. See, Connecting the Source and the Destination Nodes [page 553]</td>
<td></td>
</tr>
</tbody>
</table>
1.5.7.1.1 Creating an Instance of Content Agent

Create a service instance and a service key of content agent in your source subaccount. The service key details are needed while creating the ContentAssemblyService destination.

Context

Content Agent allows you to assemble the content of different content providers, and export it to the transport queue.

Procedure

1. Create a service instance of Content Agent. To create the service instance, follow the steps described in Create Instance.
2. Create a service key of Content Agent. For more information, see Create Service Key.
   Once the service key is created, make a note of the url, clientid, and clientsecret as these details would be needed while creating HTTP destination ContentAssemblyService for Content Agent. To copy the details, perform the following steps:
   1. Choose the Content Agent service instance that you recently created to expand the right-pane.
   2. To view the credentials, choose the Content Agent <Service Key Name>.
   3. Choose the JSON tab and copy the URL.

Results

You’ve created an instance of Content Assembly Service and its corresponding service key in the source subaccount.

1.5.7.1.2 Creating Content Assembly Service Destination

Create destination ContentAssemblyService in source subaccount to make API calls to the Content Assembly Service.

Prerequisites

Create an instance of Content Agent and fetch the service keys. For more information, see Creating an Instance of Content Agent [page 540].
Procedure

1. In your web browser, log on to SAP BTP Cockpit and navigate to your source subaccount.
2. Choose the Destinations tab in the left-hand pane.
3. Choose New Destination.
4. In Destination Configuration section, provide values in fields based on description in table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provide value as ContentAssemblyService.</td>
</tr>
<tr>
<td></td>
<td>Please note that this value is case-sensitive.</td>
</tr>
<tr>
<td>Type</td>
<td>HTTP</td>
</tr>
<tr>
<td>Description</td>
<td>You can provide a description for your reference. This field is optional.</td>
</tr>
<tr>
<td>URL</td>
<td>Provide the URL from the service key details.</td>
</tr>
<tr>
<td>Proxy Type</td>
<td>Internet</td>
</tr>
<tr>
<td>Authentication</td>
<td>OAuth2ClientCredentials</td>
</tr>
<tr>
<td>Client ID</td>
<td>Provide the client ID from the service key details.</td>
</tr>
<tr>
<td>Client Secret</td>
<td>Enter the client secret.</td>
</tr>
<tr>
<td>Token Service URL</td>
<td>Provide the URL from the service key details.</td>
</tr>
<tr>
<td></td>
<td>i Note: Append the value oauth/token to the Token Service URL.</td>
</tr>
<tr>
<td>Token Service User</td>
<td>No inputs required.</td>
</tr>
<tr>
<td>Token Service Password</td>
<td>No inputs required.</td>
</tr>
</tbody>
</table>

5. Choose Save.

You can also do a Check Connection to verify whether you’ve added the destination correctly. Once you perform a check connection, the following pop-up message appears: Connection to "ContentAssemblyService" is established. Response returned: "401: Unauthorized"

i Note
In this case, 401 is an accepted response code.
Results

You’ve created the destination ContentAssemblyService.

1.5.7.1.3 Creating an Instance of API portal, API Management

Create an instance of API portal, API Management in your source subaccount and create a service key. The service key details are needed while creating the APIManagement destination.

Prerequisites

Create an API portal, API Management subaccount and subscribe to it. Set it as your source subaccount, from where you can start exporting the API Management content. For more information, see Setting Up API Portal Application [page 43].

Procedure

1. Create a service instance for API Management, API portal.
   Follow the steps described in "Creating a Service Instance in the API Management, API portal" section in API Access plan for API Portal [page 56].

   i Note
   If you don’t see the API Management, API portal instance under the Instances tab on the Instances and Subscription page, then you must add the entitlement.

   To add the entitlement:
   1. Choose the Entitlements tab from the left pane and choose Configure Entitlements ➤ Add Service Plans ➤ Add Service Plan.
   2. Under Entitlements available for this subaccount, select API Management, API portal.
   3. On the right pane, select the available plans for API Management, API portal and choose Add ( ) Service Plan.
   4. On the Entitlements page, choose Save.

2. Create a service key for API portal, API Management.
   Follow the steps described in "Creating a Service Key" section in API Access plan for API Portal [page 56].
i Note

While configuring JSON, only the **APIportal.Administrator** role must be assigned to you and not the "APIPortal.Guest" and "APIManagement.SelfService.Administrator" roles as mentioned in API Access plan for API Portal [page 56].

Once the service key is created, make a note of the url, clientid, clientsecret, and token as these details would be needed while creating HTTP destination **APIManagement**. To copy the details, perform the following steps:

1. Choose the API portal, API Management service instance that you created recently, to expand the right-pane.
2. To view the credentials, choose the `<Service Key Name>`.
3. Choose the **JSON** tab and copy the details.

**Sample Code**

```json
{
    "url": "https://<apiportal application name>.cfapps.sap.hana.ondemand.com",
    "tokenUrl": "https://<Space name>.authentication.sap.hana.ondemand.com/oauth/token",
    "clientId": "sb-apiallexxxxxxxxx!xxxx|api-portal-xsuaa!bxxxx",
    "clientSecret": "xxxxxxxxxxxxxxxxxxxxxxx=
}
```

**Results**

You’ve created an instance and a service key of API portal, API Management in the source subaccount.

**1.5.71.4 Creating API Management Destination**

Create destination **APIManagement** in your source subaccount to make API calls for fetching the API content from the API portal workspace.

**Prerequisites**

Create an instance of API portal, API Management service and fetch the service keys from the service instance as shown in the samplecode. For more information, see Creating an Instance of API portal, API Management [page 542].

**Sample Code**

```json
{
    "url": "https://<apiportal application name>.cfapps.sap.hana.ondemand.com",
```
Procedure

1. In your web browser, log on to SAP BTP Cockpit and navigate to your source subaccount.
2. Choose the Destinations tab in the left-hand pane.
3. Choose New Destination.
4. In Destination Configuration section, provide values in fields based on description in table.

**i Note**

Use the Client ID, Client Secret, and the Token Service URL from the Prerequisite section.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter APIManagement as the destination name. Please note that this value is case-sensitive.</td>
</tr>
<tr>
<td>Type</td>
<td>Enter HTTP as the supported type.</td>
</tr>
<tr>
<td>Description</td>
<td>Enter a brief description stating the purpose of creating a new destination in the Description field.</td>
</tr>
<tr>
<td>URL</td>
<td>Provide the URL from the service key details and append /api/1.0/transportmodule/Transport to it.</td>
</tr>
<tr>
<td>Proxy Type</td>
<td>Internet</td>
</tr>
<tr>
<td>Authentication</td>
<td>Select the authentication type as OAuth2ClientCredentials.</td>
</tr>
<tr>
<td>Client ID</td>
<td>Provide the client ID from the service key details.</td>
</tr>
<tr>
<td>Client Secret</td>
<td>Enter the client secret.</td>
</tr>
<tr>
<td>Token Service URL</td>
<td>Provide the URL from the service key details.</td>
</tr>
</tbody>
</table>

5. Choose Save.

You can also do a Check Connection to verify whether you've added the destination correctly. Once you perform a check connection, the following pop-up message appears: Connection to "APIManagement" is established.
Results

You've created the destination *APIManagement*.

1.5.7.1.5 Creating an Instance of SAP Cloud Transport Management Service

Create a service instance and a service key of SAP Cloud Transport Management service in your transport subaccount. The service keys details are needed while creating the *TransportManagement* destination.

Prerequisites

Create a *Transport* subaccount and subscribe to SAP Cloud Transport Management service. Set up and subscribe to SAP Cloud Transport Management service as described in *Set Up the Environment to Transport Content Archives directly in an Application*.

To view and access the SAP Cloud Transport Management service, assign TMS_ADMIN and TMS_VIEWER roles to yourself. To set the roles, scroll down to "Steps to Assign User Roles and Permissions" section in *Set Up the Environment to Transport Content Archives directly in an Application*.

Procedure

1. Create a service instance for SAP Cloud Transport Management service.
   To create the service instance, follow step 9 described in *Set Up the Environment to Transport Content Archives directly in an Application*.

   ![i Note]
   In case you don't see the *Cloud Transport Management*, instance under the *Instances* tab on the *Instances and Subscription* page, then you must add the entitlement.

   To add the entitlement:
   1. Choose the *Entitlements* tab from the left pane and choose *Configure Entitlements* >> *Add Service Plans*.
2. Under *Entitlements available for this subaccount*, search and select *Cloud Transport Management*.
3. On the right pane, select the available plans for Cloud Transport management and choose *Add ( ) Service Plan*.
4. On the *Entitlements* page, choose *Save*.

2. Create a service key for SAP Cloud Transport Management service. To create the service key, follow step 10 described in *Set Up the Environment to Transport Content Archives directly in an Application*.

Once the service key is created, make a note of the url, clientid, clientsecret, and token as these details would be needed while creating HTTP destination *TransportManagement*. To copy the details, perform the following steps:
1. Choose the service instance *<API Portal, API Management>* that you created recently, to expand the right-pane.
2. To view the credentials, choose the *<Service Key Name>*.
3. Choose the *JSON* tab and copy the details.

```
{ 
  "uaa": { 
    "clientid": "sb-ebxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx!bxxxx|alm-ts-backend!bxxx", 
    "clientsecret": "xxxxxxxxxxxxxxxxxxxxxxxxx=",
    "url": "https://<Space name>.authentication.sap.hana.ondemand.com",
    "identityzone": "<Space name>",
    "identityzonid": "xxxxx-xxxx-xxxx-xxxx-xxxxx",
    "tenantid": "xxxxx-xxxx-xxxx-xxxxx",
    "tenantmode": "dedicated",
    "sburl": "https://internal-xsuaa.authentication.sap.hana.ondemand.com",
    "apiurl": "https://api.authentication.sap.hana.ondemand.com",
    "verificationkey": "-----BEGIN PUBLIC KEY-----xxxxxxxxxx-----END PUBLIC KEY-----",
    "xsappname": "xxxxx-xxxx-xxxx-xxxxx!bxxxx|alm-ts-backend!bxxx", 
    "uaadomain": "authentication.sap.hana.ondemand.com",
    "zoneid": "xxxxx-xxxx-xxxx-xxxxx"
  }, 
  "uri": "https://transport-service-app-backend.ts.cfapps.sap.hana.ondemand.com"
}
```

**Results**

You’ve created an instance and a service key of SAP Cloud Transport Management service in the transport subaccount.
1.5.7.1.6 Adding a Source Node in Transport Management Applications

The Transport Management Application contains the transported content, so you need a Source node to represent the source endpoint.

Context

To add a Source node in the Transport Management Applications, execute the following steps:

Procedure

1. In your web browser, log on to SAP BTP Cockpit and navigate to your Transport subaccount.
2. Choose Instances and Subscriptions from the left pane.
3. Go to Subscriptions Cloud Transport Management and choose Go To Application.
4. Choose Transport Nodes from the left pane.
5. Choose + to add a source node.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the user has a Cloud Integration subscription in the same source subaccount and is opting for SAP Cloud Transport Management service for transporting the content, then the source node can be reused for the Integration suite.</td>
</tr>
<tr>
<td>If Cloud Integration and API Management capabilities are activated under Integration suite subscription, then both the capabilities can use the same source node and the destination node.</td>
</tr>
</tbody>
</table>

6. In the Create Node dialog box, enter a node name that is unique, for example Source_node1, and enable the Allow Upload to Node option.
7. Choose OK.

Results

You’ve added a Source node in the Cloud Transport Management Applications.
1.5.7.1.7 Creating Transport Management Destination

Create destination `TransportManagement` in your source subaccount to make API calls to the SAP Cloud Transport Management service (TMS).

**Prerequisites**

Create an instance of SAP Cloud Transport Management service and fetch the service keys from the service instance as shown in the sample code. For more information, see Creating an Instance of SAP Cloud Transport Management Service [page 545].

```json
```

**Procedure**

1. In your web browser, log on to SAP BTP Cockpit and navigate to your source subaccount.
2. Choose the Destinations tab in the left-hand pane.
3. Choose New Destination.
4. In Destination Configuration section, provide values in fields based on description in table.

**Note**

Use the Client ID, Client Secret, and the Token Service URL from the Prerequisite section.
<table>
<thead>
<tr>
<th>Fields</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter <code>TransportManagementService</code> as the destination name.</td>
</tr>
<tr>
<td>Type</td>
<td>Enter <code>HTTP</code> as the supported type.</td>
</tr>
<tr>
<td>Description</td>
<td>Enter a brief description stating the purpose of creating a new destination in the <code>Description</code> field.</td>
</tr>
<tr>
<td>URL</td>
<td>Provide the URL from the service key details of the SAP Cloud Transport Management service plan.</td>
</tr>
<tr>
<td>Proxy Type</td>
<td>Internet</td>
</tr>
<tr>
<td>Authentication</td>
<td>Select the authentication type as <code>OAuth2ClientCredentials</code>.</td>
</tr>
<tr>
<td>Client ID</td>
<td>Provide the client ID from the service key details of the SAP Cloud Transport Management service plan.</td>
</tr>
<tr>
<td>Client Secret</td>
<td>Enter the client secret.</td>
</tr>
<tr>
<td>Token Service URL</td>
<td>Provide the Token Service URL from the service key details of the SAP Cloud Transport Management service plan.</td>
</tr>
</tbody>
</table>

5. Add an additional property to the SAP Cloud Transport Management service by choosing Edit New Property.

Add `sourceSystemId` in the first text box.

**i Note**

While entering the `sourceSystemId` in the first text box, don’t change the case of the text `sourceSystemId`.

Enter the source node that you created in the Transport Management Applications in the second text box. See Adding a Source Node in Transport Management Applications [page 547] for the steps on how to add a Source node.

6. Choose Save.

You can also do a Check Connection to verify whether you’ve added the destination correctly. Once you perform a check connection, the following pop-up message appears: Connection to "TransportManagementService" is established.

**i Note**

In case you receive a “401: Unauthorized” response code, please note that its an accepted response code.
1.5.7.1.8  Create a Deploy Service Destination in Cloud Transport Management Service Subaccount

Create a destination **TMSDeploy** in Transport subaccount for the deploy service. The deploy service calls the API Management in the destination subaccount to transport the API content into the API Management workspace.

**Prerequisites**

- You’ve already created a Destination subaccount and have subscribed to API portal.
- Create a **Space** in Destination subaccount by:
  1. Choosing **Create Space**.
  2. Choose a name and assign **Space Manager** and **Space Developer** roles to the user.
- Ensure that API portal API Access Plan is enabled for the API Management Instance creation in the Destination subaccount.

**Context**

Import request is delegated to deploy-service using destination D4.

**Procedure**

1. In your web browser, log on to SAP BTP Cockpit and navigate to Transport subaccount.
2. Choose the **Destinations** tab in the left-hand pane.
3. Choose **New Destination**.
4. In **Destination Configuration** section, provide values in fields based on description in table:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a unique name for this destination, for example, <strong>TMSDeploy</strong>.</td>
</tr>
</tbody>
</table>

You’ve created the destination *TransportManagement*. 
### Fields

<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>URL</td>
</tr>
<tr>
<td>Proxy Type</td>
</tr>
<tr>
<td>Authentication</td>
</tr>
<tr>
<td>User ID</td>
</tr>
<tr>
<td>Password</td>
</tr>
</tbody>
</table>

5. Choose **Next**. Choose **Check Connection** to verify whether you’ve added the destination correctly. Once you perform a check connection, the following pop-up message appears: **Connection to "TMSDeploy" is established.**

### Results

You’ve created the destination D4 TMS_Destination_DeployService. You’ve also created the target node Destination_node1.
1.5.7.1.9 Adding a Destination Node in Cloud Transport Management Applications

The Transport Management Application contains the transported content, so you need a Destination node to represent the target endpoint.

Context

To add a Destination node in the Cloud Transport Management Applications, execute the following steps:

Procedure

1. In your web browser, log on to SAP BTP Cockpit and navigate to your Transport subaccount.
2. Choose Instances and Subscriptions from the left pane.
3. Go to Subscriptions → Cloud Transport Management and choose Go To Application.
4. Choose Transport Nodes from the left pane.
5. Choose + to add a destination node.

   i Note

   If the user has a Cloud Integration subscription in the same source subaccount and is opting for SAP Cloud Transport Management service for transporting the content, then the destination node can be reused for the Integration suite.

   If Cloud Integration and API Management capabilities are activated under Integration suite subscription, then both the capabilities can use the same source node and the destination node.

6. In the Create Node dialog box, enter a node name that is unique, for example Destination_node1, and enable the Allow Upload to Node option.
7. Choose Multi-Target Application from the Content Type dropdown.
8. Choose the destination TMSDeploy you created in the Transport subaccount from the Destination dropdown. This destination contains the details of the destination subaccount’s org and space name. Refer Create a Deploy Service Destination in Cloud Transport Management Service Subaccount [page 550] for the Destination name.
9. Choose OK.

Results

You've added a Destination node in the Transport Management Applications.
1.5.7.1.10 Connecting the Source and the Destination Nodes

Create a transport route to connect the source tenant to the destination tenant.

Prerequisites

- You have configured source and destination nodes. For more information, see Creating Transport Management Destination [page 548] and Create a Deploy Service Destination in Cloud Transport Management Service Subaccount [page 550].
- Administrator or LandscapeOperator roles must be assigned to the user who has subscribed to the TMS application. For more information, see Security

Procedure

1. Choose Transport Routes from the left pane.
2. Choose + to add a transport route.
3. Enter the name of the transport route, and a description.
4. Choose the source and the destination nodes from the existing transport nodes.

   Note
   This destination node is created for the destination in the Transport subaccount and contains the details of the destination subaccount’s org and space name.

5. Choose OK.

Results

The transport route is created. You’ve established a connection between the source and the destination node.
1.5.7.2 Triggering Content Transport Using SAP Cloud Transport Management Service

After configuring the system for transport, you can start transporting the API proxy and the API artifacts from the source to the destination API portal.

Context

When transport is triggered, the API proxy and its related artifacts, such as API provider, Key Store Certificate, Trust Store, and Key Value Maps is transported to the destination. However, you can also trigger the transport of each of these artifacts individually.

Transporting an API Proxy from Source to Destination [page 554]
When transport is triggered for an API proxy, all artifacts of the API proxy get transported along with the API.

Transporting an API Provider from Source to Destination [page 556]
You can choose to transport a single API provider from the source to the destination API portal. When you transport an API provider, it gets created in the destination.

Transporting a Certificate from Source to Destination [page 557]
You can choose to transport a single Key Store Certificate or a Trust Store Certificate from the source API portal to the destination API portal.

Transporting a Key Value Map from Source to Destination [page 558]
You can transport a single Key Value Map from the source API portal to the destination API portal.

Transporting a Product from Source to Destination [page 559]
You can choose to transport a single Product from the source to the destination API portal. When you transport a Product, it gets created in the destination.

Editing the Security Fields for the Imported API Entities [page 561]
To use the imported API entities, the security fields for these entities, which carried dummy values during the import due to security reasons must be replaced with the actual values.

1.5.7.2.1 Transporting an API Proxy from Source to Destination

When transport is triggered for an API proxy, all artifacts of the API proxy get transported along with the API.

Context

API proxies always get imported to the destination API portal in the deployed state.
**i Note**

If the API proxy and its artifacts already exist in the destination, only the API proxy gets overwritten during transport. All other artifacts of the API proxy, such as API provider, Key Store Certificate, Trust Store, and Key-Value Maps remain unaffected.

**Procedure**

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose *Develop*.
3. Choose the API that you want to transport on the *APIs* tab page.
4. Choose the *Action* icon against the required API and then select the *Transport* option. Alternatively, you can open the required API and in the details page select the option *Transport*.
5. On the *Transport* popup, provide a description and choose *Yes*.
   The reference number for the transport request gets generated.
   In the Transport workbench, you can search for this API in the destination node under the *Transport Description*.
6. Go to the Transport subaccount and perform the following steps to navigate to the *Transport Nodes*.
   a. In your web browser, log on to SAP BTP Cockpit and navigate to your Transport subaccount.
   b. Choose *Instances and Subscriptions* from the left pane.
   c. Go to *Subscriptions ➤ Cloud Transport Management ➤* and choose *Go To Application*.
   d. Choose *Transport Nodes* from the left pane.
7. Select the destination node, which points to the destination API portal.
8. Under the *Transport Description* column of the destination node, search for the API for which you triggered the transport.
9. Choose *Import Selected* to import the selected API in the queue to the destination node.

**Results**

The API content from the source API portal is transported to the destination API portal.

**Next Steps**

Once the API content is transported to the destination API portal, you must ensure that the dummy security values that got imported along with the API entity must be replaced with the actual values. For more information, see *Editing the Security Fields for the Imported API Entities [page 561]*.
1.5.7.2.2 Transporting an API Provider from Source to Destination

You can choose to transport a single API provider from the source to the destination API portal. When you transport an API provider, it gets created in the destination.

**Context**

*Note*

While transporting the API provider, if there are any associated certificates (Key Store or Trust Store), those would get transported to the destination API portal as well.

If you transport an API provider that already exists in the destination API portal, the API provider doesn't get overwritten.

**Procedure**

1. Log on to the API portal.
2. Choose the navigation icon on the left and choose Configure.
3. Choose the API providers that you want to transport to the API Providers tab page.
4. Choose the Action icon against the required API provider and then select the Transport option.
   Alternatively, you can open the required API provider and in the details page select the option Transport.
5. On the Transport popup, provide a description and choose Yes.
   The reference number for the transport request gets generated.
   In the Transport workbench, you can search for the API provider in the destination node under Transport Description.
6. Go to the Transport subaccount and perform the following steps to navigate to the Transport Nodes.
   a. In your web browser, log on to SAP BTP Cockpit and navigate to your Transport subaccount.
   b. Choose Instances and Subscriptions from the left pane.
   c. Go to Subscriptions Cloud Transport Management and choose Go To Application.
   d. Choose Transport Nodes from the left pane.
7. Select the destination node, which points to the destination API portal.
8. Under the Transport Description column of the destination node, search for the API provider for which you triggered the transport.
9. Choose Import Selected to import the selected API provider in the queue to the destination node.
Results

Go to the API portal destination subaccount, and choose Configure. Under the API Providers tab, look for the API provider you transported. The API provider you transported appears on the list.

1.5.7.2.3 Transporting a Certificate from Source to Destination

You can choose to transport a single Key Store Certificate or a Trust Store Certificate from the source API portal to the destination API portal.

Context

**i Note**

Only the certificate with dummy content gets transported from the Source to the destination API portal.

If you transport a Certificate that already exists in the destination API portal, the Certificate doesn’t get overwritten.

Procedure

1. Log on to the Source API portal.
2. Navigate to the Configure tab on the left pane.
3. Choose the Certificate that you want to transport on the Certificates tab page.
4. Choose the Action icon against the required Certificate and then select the Transport option. Alternatively, you can open the required Certificate and in the details page select the option Transport.
5. On the Transport popup, provide a description and choose Yes.

The reference number for the transport request gets generated.

In the Transport workbench, you can search for the Certificate in the destination node under Transport Description.

6. Go to the Transport subaccount and perform the following steps to navigate to the Transport Nodes.
   a. In your web browser, log on to SAP BTP Cockpit and navigate to your Transport subaccount.
   b. Choose Instances and Subscriptions from the left pane.
   c. Go to Subscriptions Cloud Transport Management and choose Go To Application.
   d. Choose Transport Nodes from the left pane.
7. Select the destination node, which points to the destination API portal.
8. Under the Transport Description column of the destination node, search for the Certificate for which you triggered the transport.
9. Choose **Import Selected** to import the selected Certificate in the queue to the destination node.

**Results**

Go to the API portal of the Destination subaccount, and choose **Configure**. Under the **Certificates** tab, look for the Certificate you transported. The certificate with dummy content appears on the list.

### 1.5.7.2.4 Transporting a Key Value Map from Source to Destination

You can transport a single Key Value Map from the source API portal to the destination API portal.

**Context**

For encrypted Key Value Maps, the Key-Value Map is transported with dummy values from the source to the destination API portal. For Non-encrypted Key Value Maps, the Key-Value Map is transported as is.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you transport a Key Value Map that already exists in the destination API portal, the Key Value Map doesn’t get overwritten.</td>
</tr>
</tbody>
</table>

**Procedure**

1. Log on to the Source API portal.
2. Navigate to the **Configure** tab on the left pane.
3. Choose the **Key Value Map** that you want to transport on the **Key Value Maps** tab page.
4. Choose the **Action** icon against the required Key Value Map and then select the **Transport** option. Alternatively, you can open the required Key Value Map and in the details page select the option **Transport**.
5. On the **Transport** popup, provide a description and choose **Yes**. The reference number for the transport request gets generated.

In the Transport workbench, you can search for the Key Value Map in the destination node under **Transport Description**.

6. Go to the Transport subaccount and perform the following steps to navigate to the **Transport Nodes**.
   a. In your web browser, log on to SAP BTP Cockpit and navigate to your Transport subaccount.
   b. Choose **Instances and Subscriptions** from the left pane.
c. Go to Subscriptions ➤ Cloud Transport Management ➤ and choose Go To Application.
d. Choose Transport Nodes from the left pane.

7. Select the destination node, which points to the destination API portal.
8. Under the Transport Description column of the destination node, search for the Key Value Map for which you triggered the transport.
9. Choose Import Selected to import the selected Key Value Map in the queue to the destination node.

Results

Go to the API portal of the Destination subaccount, and choose Configure. Under the Key Value Maps tab, look for the Key Value Map you transported. The Key Value Map appears on the list.

1.5.7.2.5 Transporting a Product from Source to Destination

You can choose to transport a single Product from the source to the destination API portal. When you transport a Product, it gets created in the destination.

Context

APIs, Permissions, and Custom Attributes, that are associated with the Product get transported along with the Product and get created in the destination API portal. However, the Rate Plan associated with the Product doesn't get transported.

Note

If the Product and its associated entities (APIs, Permissions, and Custom Attributes) already exist in the destination API portal, they get overwritten during the transport. Additionally, the APIs attached to the Product get imported to the destination API portal in the deployed state. However, the artifacts of the APIs associated to the Product (such as API Provider, Key Store Certificate, Trust Store, and Key-Value Maps) remain unaffected if they already exist in the destination API portal.

Also, transporting a product from the source to the destination API portal in a draft state is not allowed if the product already exists in the destination API portal in a published state and vice versa.

While transporting a product that already exists in the destination portal, ensure that the product is in the same state in both the source and the destination API portal. If the product is in the same state, it gets overwritten in the destination during the transport.
Procedure

1. Log on to the Source API portal.
2. Navigate to the Develop tab on the left pane.
3. Choose the Product that you want to transport on the Products tab page.
4. Choose the Action icon against the required Product and then select the Transport option. Alternatively, you can open the required Product and in the details page select the option Transport.
5. On the Transport popup, provide a description and choose Yes.

The reference number for the transport request gets generated.

In the Transport workbench, you can search for this Product in the destination node under the Transport Description.

6. Go to the Transport subaccount and perform the following steps to navigate to the Transport Nodes.
   a. In your web browser, log on to SAP BTP Cockpit and navigate to your Transport subaccount.
   b. Choose Instances and Subscriptions from the left pane.
   c. Go to Subscriptions ▶ Cloud Transport Management ▶ and choose Go To Application.
   d. Choose Transport Nodes from the left pane.
7. Select the destination node, which points to the destination API portal.
8. Under the Transport Description column of the destination node, search for the Product for which you triggered the transport.
9. Choose Import Selected to import the selected Product in the queue to the destination node.

Results

The Product from the source API portal is transported to the destination API portal.

Next Steps

Once the Product is transported to the destination API portal, you must ensure that the dummy security values that got imported along with the API entity associated with the Product must be replaced with the actual values. For more information, see Editing the Security Fields for the Imported API Entities [page 561].
1.5.7.2.6 Editing the Security Fields for the Imported API Entities

To use the imported API entities, the security fields for these entities, which carried dummy values during the import due to security reasons must be replaced with the actual values.

Context

In the destination subaccount, for the imported API entity, replace the dummy certificate attached to the API Provider with a genuine certificate. Replace the dummy Key Value Map (KVM) values with authentic values. For different Connection types, replace the dummy values in the username and password fields with valid details.

Procedure

1. Log on to the API portal in the Destination subaccount.
2. Choose the navigation icon on the top-left, and choose Configure.
3. Replace the dummy certificate in the API Provider with a genuine certificate.
   1. Choose the API Provider and choose Connection.
   2. Make a note of the certificate name displayed under Key Store Certificate and Trust Store.
   3. Navigate back to the Configure page, and look for the certificate on the Certificates tab.
   4. Delete the certificate, and with the same name create a new certificate. For more information, see Manage Certificates [page 371].
4. Replace the dummy value in the encrypted Key Value Map that got imported along with the API proxy with authentic values:
   1. Navigate to the Configure page, and choose the Key Value Maps tab.
   2. Choose the imported Key Value Map and choose Edit.
   3. Delete the dummy value from the Value field and enter an authentic value.

   **Note**
   For nonencrypted Key Value Maps, dummy values aren’t imported. You can use the value of the nonencrypted Key Value Map as is.

5. To replace the dummy values in the Open Connector, Cloud Integration, On Premise, and Internet type of API Providers, perform the following steps:
   ○ Open Connector: Choose the API Provider, navigate to the Connection tab, and choose Edit. Replace the dummy values in the Organization Secret and User Secret fields with real values.
   ○ For Cloud Integration type:
     ○ If authentication type is selected as Basic, then remove the dummy values and add valid information in the Username and Password fields.
     ○ If authentication type is selected as OAuth2ClientCredentials, then replace the dummy values in the Client ID, Client Secret, and Token URL fields with real values.
○ If authentication type is selected as ClientCertificate, then make a note of the certificate names displayed under Key Store Certificate and Trust Store. Navigate back to the Configure page, look for the certificate on the Certificates tab. Delete the certificate, and with the same name create a new certificate.

○ In case of On Premise and Internet type of API Providers, if Authentication type is set as Basic in Catalog Service Settings, then you’ve to remove the dummy values and add valid information in the Username and Password fields.

Results

All the dummy values that got imported along with the API proxy during the transport has been replaced with authentic values. You can start using the imported API proxy.

1.6 API Documentation

This section contains additional instructions on how to effectively use and integrate with an API. The standard documentation is already available on the API Business Hub.

It provides you with a concise reference manual containing additional information required to work with various entities in the API Portal. For example, when to expect a $batch call, the format of a $batch call, information about expected headers, payload format, how to create/update single and multiple records, mandatory and optional fields, and, so on.

API Portal - API Proxy (CF) [page 562]

An API Proxy is a discrete representation of an API. It’s implemented as a set of configuration files, policies, and code snippets that rely on the resource information provided by API Management.

1.6.1 API Portal - API Proxy (CF)

An API Proxy is a discrete representation of an API. It’s implemented as a set of configuration files, policies, and code snippets that rely on the resource information provided by API Management.

Prerequisite

APIPortal/Administrator role must be assigned to you to perform the operations listed in this topic.
Sample Payload

Sample payload to create an API Proxy:

- **Url**: https://<consumer API-Portal host>:<port>/apiportal/api/1.0/Management.svc/APIProxies HTTP/1.1
- **Method**: POST
- **Content type**: application/JSON
- **If you are in the Cloud Foundry environment, fetch the bearer token**:
  - **Service url**: https://<consumer API-Portal host>/apiportal/api/1.0/Management.svc/APIProxies HTTP/1.1
  - **Method**: HEAD
  - **Request Header**: Authorization: Bearer <Token for API access>
  - **Response**: bearer-token value

To know how to retrieve this token, see API Access plan for API Portal [page 56].

Payload for url-based APIProxy:

```json
{
  "name": "sapapi",
  "version": "1",
  "title": "sapapi",
  "releaseStatus": "Active",
  "releaseMetadata": "{"reason": ""}",
  "description": "",
  "isPublished": false,
  "service_code": "REST",
  "provider_name": "NONE",
  "status_code": "REGISTERED",
  "state": "DEPLOYED",
  "proxyEndPoints": [{
    "base_path": "/sapapi",
    "name": "default",
    "isDefault": true,
    "apiResources": [],
    "conditionalFlows": [],
    "properties": [],
    "routeRules": [{
      "name": "default",
      "targetEndPointName": "default",
      "sequence": 1
    }],
    "virtualhosts": [{
      "__metadata": {
        "uri": "VirtualHosts('default')"
      }
    }],
    "targetEndPoints": [{
      "name": "default",
      "isDefault": true,
      "url": "https://sap.com",
      "relativePath": null,
      "provider_id": "NONE",
      "properties": [],
      "targetAPIProxyName": null
    }],
    "isVersioned": false,
    "__metadata": {
      "type": "apiportal.APIProxy"
    }
  }]
}``

SAP API Management
SAP API Management in the Cloud Foundry Environment
### Attributes

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Type</th>
<th>Nullable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>true</td>
<td>string</td>
<td>false</td>
<td>&quot;&quot;</td>
<td>It’s the technical name of the API Proxy and must be unique. This is a mandatory field, therefore this attribute must not be empty.</td>
</tr>
<tr>
<td>version</td>
<td>false</td>
<td>string</td>
<td>false</td>
<td>&quot;1&quot;</td>
<td>It’s a non mandatory field, and the default value is 1.</td>
</tr>
<tr>
<td>title</td>
<td>true</td>
<td>string</td>
<td>false</td>
<td></td>
<td>It’s the API title displayed on the Product. This is a mandatory field, therefore this attribute must not be empty.</td>
</tr>
<tr>
<td>isVersioned</td>
<td>false</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
<td>Checks whether the API has the version information or not.</td>
</tr>
<tr>
<td>releaseStatus</td>
<td>false</td>
<td>string</td>
<td>false</td>
<td>Active</td>
<td>Current state of the API Proxy.</td>
</tr>
<tr>
<td>releaseMetadata</td>
<td>false</td>
<td>string</td>
<td>false</td>
<td>&quot;&quot;&quot;&quot;&quot;</td>
<td>It’s the API Proxy release metadata.</td>
</tr>
<tr>
<td>description</td>
<td>false</td>
<td>string</td>
<td>false</td>
<td></td>
<td>It’s the API Proxy details.</td>
</tr>
<tr>
<td>isPublished</td>
<td>true</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
<td>Checks whether the API Proxy is published or not. This is a mandatory field, therefore this attribute must not be empty.</td>
</tr>
<tr>
<td>Field</td>
<td>Mandatory</td>
<td>Type</td>
<td>Nullable</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>---------</td>
<td>----------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>service_code</td>
<td>false</td>
<td>string</td>
<td>false</td>
<td>&quot;REST&quot;</td>
<td>Type of APIProxy. Possible values are REST, ODATA or SOAP.</td>
</tr>
<tr>
<td>provider_name</td>
<td>false</td>
<td>string</td>
<td>false</td>
<td>&quot;NONE&quot;</td>
<td>It's the technical name of the API Proxy provider.</td>
</tr>
<tr>
<td>status_code</td>
<td>true</td>
<td>string</td>
<td>false</td>
<td>REGISTERED</td>
<td>It's the API Proxy status whether it's EXTERNAL or REGISTERED. This is a mandatory field, therefore this attribute must not be empty.</td>
</tr>
<tr>
<td>state</td>
<td>true</td>
<td>string</td>
<td>false</td>
<td>UNDEPLOYED</td>
<td>It's the API Proxy deployed status on runtime. This is a mandatory field, therefore this attribute must not be empty.</td>
</tr>
</tbody>
</table>
### Field | Mandatory | Type | Nullable | Default | Description
--- | --- | --- | --- | --- | ---
proxyEndPoints | true | List&lt;APIProxyEndPoint&gt; | false | | It's the configuration of proxy endpoint based on the configuration provided in API Provider or Server metadata in Swagger JSON.

This defines the way client applications consume the APIs. You configure the ProxyEndpoint to define the URL of your API proxy. The proxy endpoint also determines whether applications access the API proxy over HTTP or HTTPS. You usually attach policies to the ProxyEndpoint to enforce security, quota checks, and other types of access control and rate-limiting.

This is a mandatory field, therefore this attribute must not be empty. For the more information on the proxyEndPoints attributes, see.

#### Sample Code

```java
[{  "base_path": "/{base_path}",  "name": "default",  "isDefault": true,  "apiResources": [],  "conditionalFlows": [],  "properties": [],  "routeRules": [{  "name": "default",  "targetEndpointName": "default",  "sequence": 1 }],  "virtualhosts": [{  "__metadata": {  "uri": "VirtualHosts(default)"  }  }]
}]
```
<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Type</th>
<th>Nullable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetEndPoints</td>
<td>true</td>
<td>List&lt;APITargetEndPoint&gt;</td>
<td>false</td>
<td></td>
<td><img src="https://sap.com" alt="Sample Code" /> It’s the configuration of the target endpoint based on the configuration provided in Swagger JSON. It defines the way the API proxy interacts with your backend services. You configure the TargetEndpoint to forward requests to the proper backend service, including defining any security settings, HTTP or HTTPS protocol, and other connection information. You can attach policies to the TargetEndpoint to ensure that response messages are properly formatted for the app that made the initial request. This is a mandatory field, therefore this attribute must not be empty. For the more information on the APITargetEndPoint attributes, see.</td>
</tr>
<tr>
<td>isUnmanaged</td>
<td>false</td>
<td>boolean</td>
<td>true</td>
<td>false</td>
<td>Checks whether the API Proxy is externally managed or not.</td>
</tr>
</tbody>
</table>

SAP API Management
SAP API Management in the Cloud Foundry Environment
<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Type</th>
<th>Nullable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isCopy</td>
<td>false</td>
<td>boolean</td>
<td>true</td>
<td>false</td>
<td>Checks whether the API Proxy to be cloned or not.</td>
</tr>
<tr>
<td>hasChanges</td>
<td>false</td>
<td>boolean</td>
<td>true</td>
<td>false</td>
<td>Checks whether the API Proxy has delta changes or not.</td>
</tr>
<tr>
<td>resources</td>
<td>false</td>
<td>List&lt;Resource&gt;</td>
<td>true</td>
<td>null</td>
<td>Describes the API Proxy attached resources.</td>
</tr>
<tr>
<td>attachments</td>
<td>false</td>
<td>List&lt;Attachment&gt;</td>
<td>true</td>
<td>null</td>
<td>Describes the API Proxy attachments</td>
</tr>
<tr>
<td>policies</td>
<td>false</td>
<td>List&lt;Policy&gt;</td>
<td>true</td>
<td>null</td>
<td>It’s the policies attached to the API Proxy.</td>
</tr>
<tr>
<td>apiProducts</td>
<td>false</td>
<td>List&lt;APIProduct&gt;</td>
<td>true</td>
<td>null</td>
<td>It’s the Products associated to the API Proxy.</td>
</tr>
<tr>
<td>metadata</td>
<td>false</td>
<td>Object</td>
<td>false</td>
<td></td>
<td>It’s the default metadata required to create an API proxy in runtime.</td>
</tr>
</tbody>
</table>

Payload for APIProvider based APIProxy with Policy:

```json
{
    "name": "RMTSAMPLEFLIGHT",
    "version": "1",
    "title": "RMTSAMPLEFLIGHT",
    "releaseStatus": "Active",
    "releaseMetadata": "{"reason":"\""},
    "description": "OData Channel - Reference SFlight Data Provider",
    "isPublished": false,
    "service_code": "ODATA",
    "provider_name": "ES5",
    "status_code": "REGISTERED",
}
```
"state": "DEPLOYED",
"proxyEndPoints": [{
  "base_path": "/RMTSAMPLEFLIGHT",
  "name": "default",
  "isDefault": true,
  "apiResources": [{
    "canShowDelete": false,
    "canShowGet": true,
    "canShowPost": true,
    "canShowPut": false,
    "canShowOption": false,
    "canShowHead": false,
    "canShowPatch": false,
    "isDeleteChecked": false,
    "isGetChecked": true,
    "isPostChecked": true,
    "isPutChecked": false,
    "isOptionChecked": false,
    "isHeadChecked": false,
    "isPatchChecked": false,
    "resource_path": "/BookingCollection",
    "title": "BookingCollection",
    "name": "BookingCollection",
    "description": "",
    "documentations": [{
      "content": "",
      "mime_type": "HTML",
      "apiResourceName": "BookingCollection",
      "locale": "en_US"
    }]
  }],
  "conditionalFlows": [{
    "name": "BookingCollection",
    "conditions": "proxy.pathsuffix MatchesPath "/BookingCollection" OR proxy.pathsuffix MatchesPath "/BookingCollection/**" OR proxy.pathsuffix MatchesPath "/BookingCollection(**")AND(request.verb = "POST" OR request.verb = "GET")",
    "readOnly": true,
    "sequence": 1
  },
  "name": "DefaultFaultFlow",
  "conditions": "proxy.pathsuffix MatchesPath "/BookingCollection" OR proxy.pathsuffix RemovesPath "/BookingCollection/**" OR proxy.pathsuffix RemovesPath "/BookingCollection(**")",
  "sequence": 2,
  "request": {
    "steps": [{
      "policy_name": "defaultRaiseFaultPolicy",
      "condition": "",
      "sequence": 1
    }
  ]
}]
}],
"properties": [],
"routeRules": [{
  "name": "default",
  "targetEndPointName": "default",
  "sequence": 1
}],
"virtualhosts": [{
  "__metadata": {
    "uri": "VirtualHosts('default')"
  }
}]
}],
"targetEndPoints": [{
  "name": "default",
  "isDefault": true,
Sample payload to update APIProxy:

- **Url:** https://<consumer API-Portal host>:<port>/apiportal/api/1.0/Management.svc/$batch HTTP/1.1
- **Method:** POST
- **If you are in the Neo environment,** fetch the x-csrf-token:
  - Request Header: x-csrf-token: fetch
- **If you are in the Cloud Foundry environment,** fetch the bearer token:
  - Request Header: Authorization: Bearer <Token for API access>
  - To know how to retrieve this token, see [API Access plan for API Portal](page 56).

Sample Code for API Proxy description update:

```sh
Sample Code

Header:
Content-Type: multipart/mixed;boundary=batch_243c-357a-0730
Payload:
--batch_243c-357a-0730
Content-Type: multipart/mixed; boundary=changeset_9657-05a6-d1fd
--changeset_9657-05a6-d1fd
Content-Type: application/http
Content-Transfer-Encoding: binary

PUT APIProxies(name='sapapi') HTTP/1.1
x-csrf-token: cec2ea9957958732-2t1J66SYh12-aZ_13Aytdhkypac
Accept-Language: en
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
Content-Type: application/json
Content-Length: 265

{"name":"sapapi","title":"sapapi","description":"<p>description added</p>","version":"1","status_code":"PUBLISHED","service_code":"REST","isPublished":false,"releaseStatus":"Active","successorAPI":null,"releaseMetadata":null,"provider_name":"NONE","state":"DEPLOYED"}
--changeset_9657-05a6-d1fd--
Sample payload to update APIProxy by adding a policy:

### Sample Code

**Header:**
```
Content-Type: multipart/mixed;boundary=batch_0213-5a91-36e0
```
```
--batch_0213-5a91-36e0
```
```
Content-Type: multipart/mixed; boundary=changeset_af75-4dba-2842
```
```
--changeset_af75-4dba-2842
Content-Type: application/http
Content-Transfer-Encoding: binary
PUT APIProxies(name='sapapi') HTTP/1.1
x-csrf-token: cec2ea9957958732-2t1J6RShh12-a7_13Aytdhkypac
Accept-Language: en
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
RequestId: f098f516-daab-44ec-8b11-7a9c8d832ac
Content-Type: application/json
Content-Length: 265
```
```
{"name":"sapapi","title":"sapapi","description":"<p>description added</p>","version":"1","status_code":"PUBLISHED","service_code":"REST","isPublished":false,"releaseStatus":"Active","successorAPI":null,"releaseMetadata":null,"provider_name":null,"state":"DEPLOYED"}
```
```
--changeset_af75-4dba-2842
Content-Type: application/http
Content-Transfer-Encoding: binary
POST Streams HTTP/1.1
x-csrf-token: cec2ea9957958732-2t1J6RShh12-a7_13Aytdhkypac
Accept-Language: en
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
RequestId: f098f516-daab-44ec-8b11-7a9c8d832ac
Content-Type: application/json
Content-Length: 190
```
```
{"steps":[]}
```
```
{"condition":null,"id":"id-1619435575158-702","policy_name":"vap","sequence":1}
```
```
"isRequest":true,"flow":[]
```
```
--changeset_af75-4dba-2842
Content-Type: application/http
Content-Transfer-Encoding: binary
POST Policies HTTP/1.1
x-csrf-token: cec2ea9957958732-2t1J6RShh12-a7_13Aytdhkypac
Accept-Language: en
Accept: application/json
MaxDataServiceVersion: 2.0
DataServiceVersion: 2.0
RequestId: f098f516-daab-44ec-8b11-7a9c8d832ac
Content-Type: application/json
Content-Length: 375
```
```
{"name":"vap","type":"VerifyAPIKey","policy_content":" <!--Specify in the APIKey element where to look for the variable containing the api key--> 
\n\n<VerifyAPIKey async='true' continueOnError='false' enabled='true' \r
\n<VerifyAPIKey async='true' continueOnError='false' enabled='true' \r
```
```
Sample payload for updating the API Proxy to apply policy templates:

**Sample Code**

Header:
- Content-Type: multipart/mixed; boundary=batch_243c-357a-0730
- Content-Type: multipart/mixed; boundary=changeset_9657-05a6-d1fd

Payload:
- PUT APIProxies(name='sapapi') HTTP/1.1
- x-csrf-token: cec2ea9957958732-2tI6RSYh12-aZ_13Aytdhkypac
- Accept-Language: en
- Accept: application/json
- MaxDataServiceVersion: 2.0
- DataServiceVersion: 2.0
- Content-Type: application/json
- Content-Length: 265

```json
{"name":"sapapi","title":"sapapi","description":"<p>description added</p>","version":1,"status_code":"PUBLISHED","service_code":"REST","isPublished":false,"releaseStatus":"Active","successorAPI":null,"releaseMetadata":{},"provider_name":"NONE","state":"DEPLOYED","policyTemplateNames":"sample_template"}
```

**Note**

The sample template you’re using in the $batch request should already exist in the API Portal.
### 1.6.1.1 proxyEndPoints Attributes (CF)

Refer the table below for the mandatory and optional fields, different field types, and the significance of the API proxy endpoints attributes.

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Type</th>
<th>Nullable</th>
<th>Default Payload</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>True</td>
<td>string</td>
<td>False</td>
<td>&quot;default&quot;</td>
<td>It’s the name of the proxy endpoint.</td>
</tr>
<tr>
<td>base_path</td>
<td>True</td>
<td>blob</td>
<td>False</td>
<td>&quot;/{base_path}&quot;</td>
<td>It’s the base path of the API proxy endpoint.</td>
</tr>
<tr>
<td>isDefault</td>
<td>True</td>
<td>string</td>
<td>False</td>
<td>&quot;&quot;</td>
<td>Checks whether the API proxy endpoint is default or not.</td>
</tr>
<tr>
<td>life_cycle</td>
<td>False</td>
<td>History</td>
<td>True</td>
<td>&quot;&quot;</td>
<td>It’s the history details of the API proxy endpoint. For example, createdAt, modifiedAt.</td>
</tr>
<tr>
<td>properties</td>
<td>False</td>
<td>List&lt;EndPointProperty&gt;</td>
<td>True</td>
<td>[]</td>
<td>It’s the properties of the API proxy endpoint.</td>
</tr>
<tr>
<td>routeRules</td>
<td>False</td>
<td>List&lt;RouteRule&gt;</td>
<td>True</td>
<td></td>
<td>Describes the route rules of the API proxy endpoint.</td>
</tr>
</tbody>
</table>

Sample Code

```json
[{  "name":  "default",  "targetEndPointName":  "default",  "sequence": 1 }]
```
### 1.6.1.2 targetEndPoint Attributes (CF)

Refer the table below for the mandatory and optional fields, different field types, and the significance of the API target endpoints attributes.

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Type</th>
<th>Nullable</th>
<th>Default Payload</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preFlow</td>
<td>False</td>
<td>FlowRule</td>
<td>True</td>
<td>null</td>
<td>It’s the pre-flow rules of the API proxy endpoint.</td>
</tr>
<tr>
<td>postFlow</td>
<td>False</td>
<td>FlowRule</td>
<td>True</td>
<td>null</td>
<td>It’s the post-flow rules of the API proxy endpoint.</td>
</tr>
<tr>
<td>faultRules</td>
<td>False</td>
<td>List&lt;FaultRule&gt;</td>
<td>True</td>
<td>[]</td>
<td>It’s the fault rules of the API proxy endpoint.</td>
</tr>
<tr>
<td>publishUrl</td>
<td>False</td>
<td>blob</td>
<td>True</td>
<td>&quot;/{base_path}&quot;</td>
<td>It’s the base path of the API proxy endpoint for Externally Managed APIs.</td>
</tr>
<tr>
<td>conditionalFlows</td>
<td>False</td>
<td>List&lt;ConditionalFlowRule&gt;</td>
<td>True</td>
<td>[]</td>
<td>It’s the conditional flow of the API proxy endpoint.</td>
</tr>
<tr>
<td>apiResources</td>
<td>False</td>
<td>List&lt;APIResource&gt;</td>
<td>True</td>
<td>[]</td>
<td>It’s the resources of the API proxy endpoint.</td>
</tr>
<tr>
<td>virtualhosts</td>
<td>False</td>
<td>List&lt;VirtualHost&gt;</td>
<td>True</td>
<td>null</td>
<td>Describes the virtual host configuration metadata of the API proxy endpoint.</td>
</tr>
</tbody>
</table>

**Attributes**

<table>
<thead>
<tr>
<th>Field</th>
<th>Mandatory</th>
<th>Type</th>
<th>Nullable</th>
<th>Default Payload</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>true</td>
<td>string</td>
<td>false</td>
<td>&quot;default &quot;</td>
<td>It’s the default name of the API target endpoint.</td>
</tr>
<tr>
<td>Field</td>
<td>Mandatory</td>
<td>Type</td>
<td>Nullable</td>
<td>Default Payload</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>------------------</td>
<td>----------</td>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>url</td>
<td>true</td>
<td>string</td>
<td>false</td>
<td>Server url or localhost</td>
<td>It’s the server url of the API target endpoint.</td>
</tr>
<tr>
<td>provider_id</td>
<td>false</td>
<td>string</td>
<td>true</td>
<td>&quot;NONE&quot;</td>
<td>It’s the Provider ID of the API target endpoint server url.</td>
</tr>
<tr>
<td>isDefault</td>
<td>true</td>
<td>Boolean</td>
<td>false</td>
<td>&quot;default &quot;</td>
<td>It’s the default status of the API target endpoint server url.</td>
</tr>
<tr>
<td>targetAPIProxy-Name</td>
<td>false</td>
<td>string</td>
<td>true</td>
<td>null</td>
<td>It’s the proxy name of the API target endpoint.</td>
</tr>
<tr>
<td>life_cycle</td>
<td>false</td>
<td>History</td>
<td>true</td>
<td>&quot;&quot;</td>
<td>It’s the history details of the API target endpoint. For example, createdAt, modifiedAt.</td>
</tr>
<tr>
<td>relativePath</td>
<td>false</td>
<td>string</td>
<td>true</td>
<td>null</td>
<td>It’s the relative path of the API target endpoint.</td>
</tr>
<tr>
<td>properties</td>
<td>false</td>
<td>List&lt;EndPoint-Property&gt;</td>
<td>true</td>
<td>[]</td>
<td>It’s the properties of the API target endpoint.</td>
</tr>
<tr>
<td>preFlow</td>
<td>false</td>
<td>FlowRule</td>
<td>true</td>
<td>null</td>
<td>Describes the pre-flow rules of the API target endpoint.</td>
</tr>
<tr>
<td>postFlow</td>
<td>false</td>
<td>FlowRule</td>
<td>true</td>
<td>null</td>
<td>Describes the post-flow rules of the API target endpoint.</td>
</tr>
<tr>
<td>faultRules</td>
<td>false</td>
<td>List&lt;FaultRule&gt;</td>
<td>true</td>
<td>[]</td>
<td>Describes the fault rules of the API target endpoint.</td>
</tr>
</tbody>
</table>
# Security

This section describes how to secure API Management applications.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Authentication</td>
<td>To access API Management application one needs to have a valid SCN user registered with API Management solution. The SCN credentials should be used for logon to API Management solution.</td>
</tr>
<tr>
<td>Authorization</td>
<td>In API Management, you provide authorization to users by assigning relevant roles. For more information on how to provide authorizations, see Assigning User Roles [page 64].</td>
</tr>
</tbody>
</table>
| Securing APIs     | Secure your APIs by referring to the following security policies supported by API Management:  
  - Basic Authentication [page 139]  
  - OAuth v2.0 [page 236]  
  - OAuth v2.0 GET [page 249]  
  - OAuth v2.0 SET [page 252]  
  - Verify API Key [page 265]  
  - SAML Assertion Policy [page 258] |
| Security Best Practices | Security policies provide information on how to control access to your APIs with OAuth, API key and other threat protection features. For more information, refer to the following policies supported by API Management:  
  - Access Control [page 94]  
  - JSON Threat Protection [page 177]  
  - XML Threat Protection [page 283]  
  - Message Validation Policy [page 263]  
  - Regular Expression Protection [page 296] |
Traffic management

API Management supports the following traffic management policies:

- Quota [page 217]
- Spike Arrest [page 234]
- Concurrent Rate Limit [page 141]

For more information on Security policies, see https://blogs.sap.com/2017/08/22/sap-cloud-platform-api-management-api-security-best-practices/

1.7.1 Data Protection and Privacy

Governments place legal requirements on industry to protect data and privacy. We provide features and functions to help you meet these requirements. In SAP API Management we store the personal data of the application developer.

**Note**

SAP does not provide legal advice in any form. SAP software supports data protection compliance by providing security features and data protection-relevant functions, such as blocking and deletion of personal data. In many cases, compliance with applicable data protection and privacy laws are not covered by a product feature. Furthermore, this information should not be taken as advice or a recommendation regarding additional features that would be required in specific IT environments. Decisions related to data protection must be made on a case-by-case basis, taking into consideration the given system landscape and the applicable legal requirements. Definitions and other terms used in this documentation are not taken from a specific legal source.

Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocking</td>
<td>A method of restricting access to data for which the primary business purpose has ended.</td>
</tr>
<tr>
<td>Business purpose</td>
<td>A legal, contractual, or otherwise justified reason for the processing of personal data. The assumption is that any purpose has an end that is usually already defined when the purpose starts.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Consent</td>
<td>The action of the data subject confirming that the usage of his or her personal data shall be allowed for a given purpose. A consent functionality allows the storage of a consent record in relation to a specific purpose and shows if a data subject has granted, withdrawn, or denied consent.</td>
</tr>
<tr>
<td>Deletion</td>
<td>Deletion of personal data so that the data is no longer available.</td>
</tr>
<tr>
<td>End of business</td>
<td>Date where the business with a data subject ends, for example the order is completed, the subscription is canceled, or the last bill is settled.</td>
</tr>
<tr>
<td>End of purpose (EoP)</td>
<td>End of purpose and start of blocking period. The point in time, when the primary processing purpose ends (e.g. contract is fulfilled).</td>
</tr>
<tr>
<td>End of purpose (EoP) check</td>
<td>A method of identifying the point in time for a data set when the processing of personal data is no longer required for the primary business purpose. After the EoP has been reached, the data is blocked and can only be accessed by users with special authorization (for example, tax auditors).</td>
</tr>
<tr>
<td>Personal data</td>
<td>Any information relating to an identified or identifiable natural person (“data subject”). An identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or one or more factors specific to the physical, physiological, genetic, mental, economic, cultural, or social identity of that natural person. Any information relating to the application developer or the Cloud Foundry developer using SAP API Management service.</td>
</tr>
<tr>
<td>Residence period</td>
<td>The period of time between the end of business and the end of purpose (EoP) for a data set during which the data remains in the database and can be used in case of subsequent processes related to the original purpose. At the end of the longest configured residence period, the data is blocked or deleted. The residence period is part of the overall retention period.</td>
</tr>
</tbody>
</table>
**Term** | **Definition**
--- | ---
Retention period | The period of time between the end of the last business activity involving a specific object (for example, a business partner) and the deletion of the corresponding data, subject to applicable laws. The retention period is a combination of the residence period and the blocking period. SAP API Management has a retention period of six months and all the data is automatically cleaned up after the retention period.

Referenced data | Any information relating to application developer’s application.

**User Consent**

Various types of customer data are processed by and stored on SAP API Management at different times. This data gets the highest level of protection, and SAP takes dedicated measures to guarantee this security level.

To comply with user consent, SAP customers should customize SAP API Management to use their own identity provider. SAP customers using the custom identity provider should ensure that the necessary mechanism for user consent is available to allow personal data (of a natural person such as a customer, contact, or account) to be collected as well as transferred to the solution.

**Read-Access Logging**

Read Access Logging (RAL) is used to monitor and log read access to sensitive data. Data may be categorized as sensitive by law, by external company policy, or by internal company policy.

SAP API Management does not store any sensitive personal data.

**Information Report**

An information report is a collection of data relating to a data subject. A data privacy specialist may be required to provide such a report or an application may offer a self-service.

The only personal data of data subjects stored in the SAP API Management API portal is the user ID. This user ID is stored whenever a user creates an artifact on the API portal, for example an API Proxy or API Product, and this user ID can be obtained (read) only from that artifact.

To enable data subjects to obtain information about their personal data in the SAP API Management Developer Portal, we provide the following service to retrieve their personal information:

- **Service to View User Details on API business hub enterprise [page 580]**
Erasure

When handling personal data, consider the legislation in the different countries where your organization operates. After the data has passed the end of purpose, regulations may require you to delete the data. However, additional regulations may require you to keep the data longer. During this period you must block access to the data by unauthorized persons until the end of the six months retention period, when the data is finally deleted.

Personal data can also include referenced data. The challenge for deletion and blocking is to first handle referenced data and then other data, such as business partner data.

SAP API Management stores the API portal administrator’s user ID. Storing the user ID is a business requirement and the user ID is deleted when the resources created by the API portal administrator are removed.

SAP API Management stores personal information such as first name, last name, user ID, and e-mail ID of users who have logged on to the Developer Portal. All the personal information stored in the application is deleted when the access for the corresponding user is revoked.

In SAP API Management, application developers can contact their developer portal administrators to have their personal data erased and access revoked. For more information, see Revoke Access [page 507] and .

Change Log

For auditing purposes or for legal requirements, changes made to personal data should be logged, making it possible to monitor who made changes and when.

SAP API Management does not allow users to make any changes to their personal data via the API Management application. However, changes made in the identity provider are synced to the API Management application and the sync operation is logged by API Management. Changes made in the identity provider should be logged by the identity provider.

1.7.1.1 Service to View User Details on API business hub enterprise

SAP API Management allows user to view their personal data stored in the API business hub enterprise.

Service to view personal data on API business hub enterprise:

- URL: https://<Dev-Portal-URL>/api/1.0/user
- Method: GET
- Response: Fetches your personal details stored in API business hub enterprise.

```json
{
    "Name": "<user ID>
}
```
1.7.1.2 Auditing and Logging Information

Here you can find a list of the security events that are logged by TECHNICAL COMPONENT.
Security events written in audit logs

<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Proxy</td>
<td>Create API Proxy</td>
<td>• action: create</td>
<td>Create an API Proxy [page 390]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: PROXY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
{"uuid": "<msg GUID>",
 "user": "<user email>",
 "time": "2021-06-09T10:34:32.369Z",
 "id": 
 "b370c7ed-a995-4ac8-b207-9ba59f470e55",
 "success": true,
 "object": 
 {"type": "PROXY"},
 "id": 
 {"class": "com.sap.apimgmt.asmprov.core.processor.APIProxyProcessor"},
 "objectId": "PROXY",
 "action": "create",
 "message": "<based on the event the corresponding message will be logged>"}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Proxy</td>
<td>Update API Proxy</td>
<td>• action: update</td>
<td>Edit an API Proxy [page 459]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: PROXY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
"\"uuid\": <msg GUID>,
"\"user\": <user email>,
"\"time\": 2021-06-09T10:34:32.369Z,
"\"id\": <ID>,
"\"success": true,
"\"object\": {
  "\"type\": "PROXY",
  "\"id\": {
    "\"class\": "com.sap.apiManagement.asmprov.core.processor.APIProxyProcessor",
    "\"objectId\": "PROXY",
    "\"action\": "update",
    "\"message\": "<based on the event the corresponding message will be logged>"}}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Proxy</td>
<td>Delete API Proxy</td>
<td>- action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- objectType: PROXY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
{
  "uuid": "<msg GUID>",
  "msg GUID": "<user>",
  "user": "<user email>",
  "time": "2021-06-09T10:34:32.369Z",
  "id": "<ID>",
  "success": true,
  "object": {
    "type": "PROXY",
    "id": {
      "class": "com.sap.apimgmt.asmprov.core.processor.APIProxyProcessor",
      "objectId": "PROXY",
      "action": "delete",
      "message": "<based on the event the corresponding message will be logged>"
    }
  }
}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Product</td>
<td>Create API Product</td>
<td>• action: create</td>
<td>Create a Product [page 482]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: APIPRODUCT</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
{ "uuid": "<msg GUID>", "user": "<user email>", "time": "2021-06-09T10:34:32.369Z", "id": "<ID>" success: true, object: { "type": "APIPRODUCT", "id": { "class": "com.sap.apimgmt.asmprov.core.processor.APIProductProcessor", "objectId": "APIPRODUCT", "action": "create", "message": "<based on the event the corresponding message will be logged>" } |
```

SAP API Management
SAP API Management in the Cloud Foundry Environment

PUBLIC 585
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Product</td>
<td>Update API Product</td>
<td>• action: update</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: APIPRODUCT</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
{"uuid": "<msg GUID>",
"<msg GUID>": ",
"user": ",
"user": ",
"time": ",
"2021-06-09T10:34:32.369Z",
"id": ",
"<ID>",
"success": ":true,
":true,
"object": ",
"<type>": ",
"<APIPRODUCT",
"id": ",
"<APIPRODUCT",
"class": ",
"com.sap.apimgmt.asmprov.core.processor.APIProductProcessor",
"objectId": ",
"<APIPRODUCT",
"action": ",
"action": ",
"update": ",
"message": ",
"<based on the event the corresponding message will be logged>"}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Product</td>
<td>Delete API Product</td>
<td>• action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: APIPRODUCT</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
"\"uuid\":",
"<msg GUID>",
"\"user\":",
"<user e-mail>",
"<time>",
"2021-06-09T10:34:32.369Z",
"<id>",
"<ID>",
"<ID>",
"success",
":true,
"object",
"type",
"APIPRODUCT",
"id",
"<ID>",
"<ID>",
"class",
"com.sap.apimgmt.asmprov.core.processor.APIProductProcessor",
"objectId",
"APIPRODUCT",
"action",
":delete",
"message",
"<based on the event the corresponding message will be logged>"
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| Application    | Create Application Name | • action: create  
• objectType: APPLICATION | Create an Application [page 511] |

Audit Log

Example:

```json
{"uuid": "<msg GUID>",  
"msg GUID": 
", 
"user": 
"<user email>",  
"user email": 
"<time": 
"2021-06-09T10:34:32.369Z 
"," 
"id": 
"<ID> 
" 
"success 
":true, 
"object": 
{"type": 
"APPLICATION 
", 
"id": 
{"class": 
"com.sap.api.mgmt.asmprov.core.processor.ApplicationProcessor",  
"objectID": 
"APPLICATION 
"," 
"action": 
"create", 
"message": 
"<based on the event the corresponding message will be logged>"}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Update Application Name</td>
<td>• action: update</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: APPLICATION</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
{"uuid": "<msg GUID>", "user": "<user email>", "time": "2021-06-09T00:34:32.369Z", "id": "<ID>", "success": true, "object": {"type": "APPLICATION", "id": {"class": "com.sap.apimgmt.asmprov.core.processor.ApplicationProcessor"}, "objectId": "APPLICATION", "action": "update", "message": "<based on the event the corresponding message will be logged>"}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Delete Application Name</td>
<td>• action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: APPLICATION</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
"{"uuid": "<msg GUID>",
"<msg GUID>","user":"<user email>",
"<user email>","time": "2021-06-09T10:34:32.369Z",
"time": "2021-06-09T10:34:32.369Z",
"id": "<ID>",
"id": "<ID>",
"success": true,
"success": true,
"object": {
"object": {
"type": "APPLICATION",
"type": "APPLICATION",
"id": "",
"id": "",
"class": "com.sap.apiManagement.asmprov.core.processor.ApplicationProcessor",
"class": "com.sap.apiManagement.asmprov.core.processor.ApplicationProcessor",
"objectId": "APPLICATION",
"objectId": "APPLICATION",
"action": "delete",
"action": "delete",
"message": "<based on the event the corresponding message will be logged>"
"message": "<based on the event the corresponding message will be logged>"
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Provider</td>
<td>Create API Provider</td>
<td>• action: create</td>
<td>Create an API Provider [page 374]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: API PROVIDER</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
{ "uuid": "<msg GUID>",
"user": {
"email": "<user email>",
"time": "2021-06-09T10:34:32.369Z",
"id": "<ID>",
"success": true,
"object": {
"type": "API PROVIDER",
"class": "com.sap.apimng.asmprov.core.processor.APIProviderProcessor",
"objectId": "API PROVIDER",
"action": "create",
"message": "<based on the event the corresponding message will be logged>"} }```

<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Provider</td>
<td>Update API Provider</td>
<td>• action: update</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: APIPROVIDER</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```json
{"uuid": "<msg GUID>", "msg GUID": "<msg GUID>", "user": "<user email>", "user email": "<user email>", "time": "2021-06-09T16:34:32.369Z", "id": "<ID>", "success": true, "object": {"type": "APIPROVIDER", "APIPROVIDER": "", "id": "", "class": "com.sap.apimgmt.asmprov.core.processor.APIProviderProcessor", "objectID": "APIPROVIDER", "action": "update", "message": "<based on the event the corresponding message will be logged>"}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Provider</td>
<td>Delete API Provider</td>
<td>• action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: APIPROVIDER</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```json
{
  "uuid": "<msg GUID>",
  "msg GUID": "<user email>",
  "user": "<user email>",
  "time": "2021-06-09T10:34:32.369Z",
  "id": "b370c7ed-a995-4ac8-b207-9ba59f470e55",
  "success": true,
  "object": {
    "type": "APIPROVIDER",
    "id": {
      "class": "com.sap.apimgmt.asmprov.core.processor.APIProviderProcessor",
      "objectId": "APIPROVIDER",
      "action": "delete",
      "message": "<based on the event the corresponding message will be logged>"
    }
  }
}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| Developer      | Create Developer      | • action: create  
• objectType: DEVELOPER            | Onboard an Application Developer [page 504] |

Audit Log

Example:

```json
{
"uuid" : "<msg GUID>",
"user" : "<user email>",
"time" : "2021-06-09T10:34:32.369Z",
"id" : "<ID>",
"success" : true,
"object" : {
"type" : "DEVELOPER",
"id" : {
"class" : "com.sap.apimgmt.asmprov.core.processor.DeveloperProcessor",
"objectId" : "DEVELOPER",
"action" : "create",
"message" : "<based on the event the corresponding message will be logged>"
}
}
```

"Output Code"
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer</td>
<td>Update Developer</td>
<td>• action: update</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: DEVELOPER</td>
<td></td>
</tr>
</tbody>
</table>

**Audit Log:**

Example:

```json
"\"uuid\": "<msg GUID>",
"\"user\": "<user email>",
"\"time\": "2021-06-09T00:34:32.369Z",
"\"id\": "<ID>",
"\"success\": true,
"\"object\": {
"\"type\": "PROXY",
"\"id\": {
"\"class\": "com.sap.apimgmt.apimprov.core.processor.DeveloperProcessor",
"\"objectId\": "APPLICATION",
"\"action\": "update",
"\"message\": "<based on the event the corresponding message will be logged>"}}
```

--

**Output Code**

```json
"\"uuid\": "<msg GUID>",
"\"user\": "<user email>",
"\"time\": "2021-06-09T00:34:32.369Z",
"\"id\": "<ID>",
"\"success\": true,
"\"object\": {
"\"type\": "PROXY",
"\"id\": {
"\"class\": "com.sap.apimgmt.apimprov.core.processor.DeveloperProcessor",
"\"objectId\": "APPLICATION",
"\"action\": "update",
"\"message\": "<based on the event the corresponding message will be logged>"}}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer</td>
<td>Delete Developer</td>
<td>• action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: DEVELOPER</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```json
{
  "uuid": "<msg GUID>",
  "msg GUID": "",
  "user": "<user email>",
  "time": "2021-06-09T00:34:32.369Z",
  "id": "<ID>",
  "success": true,
  "object": {
    "type": "DEVELOPER",
    "id": {
      "class": "com.sap.apimgmt.asmprov.core.processor.DeveloperProcessor",
      "objectID": "<DEVELOPER>",
      "action": "delete",
      "message": "<based on the event the corresponding message will be logged>"
  }
}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>Create Environment</td>
<td>• action: create</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: ENVIRONMENT</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```json
{"uid": "<msg GUID>",
"msg GUID": ","user": ","user": "<user e-mail>",
"time": "2021-06-09T10:34:32.369Z",
"id": "<ID>",
"success": true,
"object": {
"type": "ENVIRONMENT",
"id": {
"class": "com.sap.apimgmt.asmprov.mo
del.listener.EnvironmentEntityListener",
"objectId": "ENVIRONMENT",
"action": "delete",
"delete": "",
"message": "<based on the event the corresponding meassage will be logged>"
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>Delete Environment</td>
<td>• action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: ENVIRONMENT</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```
{ "uuid": "<msg GUID>", "<user email>": "<user email>", "time": "2021-06-09T00:34:32.369Z", "id": "<ID>", "success": true, "object": { "type": "ENVIRONMENT", "id": { "class": "com.sap.amsg.asmprov.mo.del.listener.EnvironmentEntityListener", "objectId": "ENVIRONMENT", "action": "delete", "message": "<based on the event the corresponding message will be logged>" } }
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>XProperty</td>
<td>Create XProperty</td>
<td>• action: create</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: XPROPERTY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```json
{"uuid":"<msg GUID>",
"<msg GUID>":
""<user e-mail>",
"time":
"2021-06-09T10:34:32.369Z",
"id":"<ID>",
"success":true,
"object":{
"type":
"XPROPERTY",
"id":{
"class":
"com.sap.apimgmt.asmprov.model.listener.XPropertyEntityListener",
"objectId":
"XPROPERTY",
"action":
"create",
"message":
"<based on the event the corresponding message will be logged>"
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>XProperty</td>
<td>Delete XProperty</td>
<td>• action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: XPROPERTY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```json
{
    "uuid": "<msg GUID>",
    "msg GUID": "<user@email>",
    "time": "2021-06-09T10:34:32.369Z",
    "id": "<ID>",
    "success": true,
    "object": {
        "type": "XPROPERTY",
        "id": {
            "class": "com.sap.apimgmt.asmprov.mo.del.listener.XPropertyEntityListener",
            "objectId": "XPROPERTY",
            "action": "delete",
            "message": "<based on the event the corresponding message will be logged>"
        }
    }
}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTenantProperty</td>
<td>Create XTenantProperty</td>
<td>• action: create</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: XTENANT-PROPERTY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```json
{"uuid": "<msg GUID>",
 "<msg GUID>",
 "<user>":
 "<user email>",
 "time":
  "2021-06-09T10:34:32.369Z",
 "id": "<ID>",
 "success": true,
 "object": {
 "type": "XPROPERTY",
 "id": {
  "class": "com.sap.apiManagement.asmprov.model.listener.XTenantEntityListener",
  "objectId": "XTENANTPROPERTY",
  "action": "create",
  "message": "<based on the event the corresponding message will be logged>
"}}}
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTenantProperty</td>
<td>Delete XTenantProperty</td>
<td>• action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: X TENANT-PROPERTY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log:

Example:

```
"\"uuid\":\"<msg GUID>\",
"\"user\":\"<user e-mail>\",
"\"time\":\"2021-06-09T10:34:32.369Z\",
"\"id\":\"<ID>\",
"\"success\":true,
"\"object\":{
"\"type\":\"XTENANTPROPERTY\",
"\"id\":{
"\"class\":\"com.sap.apimgmt.asmprov.mo.del.listener.XTenantPropertyEntityListener\",
"\"objectId\":\"XTENANTPROPERTY\",
"\"action\":\"delete\",
"\"message\":\"<based on the event the corresponding message will be logged>\"}
```

{\"Output Code\n
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID</td>
<td>Create Application ID</td>
<td>• action: create</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: APPLICATION</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
"message": 
"\"uuid\": 
"<msg GUID>
", 
"user": 
"<user email>
", 
"time": 
"2021-07-13T05:58:25.864Z 
", 
"id": 
"<msg GUID>
", 
"object": 
{"type": 
"user", 
"id 
": 
"class": 
"com.sap.it.s pc.apimgmt.entityhandlers.Do mainHook", 
"objectId": 
<Application 
"action": 
"aboutToUpdate 
", 
"message": 
"application. update", 
"user": 
"<user email>
"}, 
"attributes 
": 
[{
"Application Id": 
"value": 
"status": 
"BEGIN", 
"category": 
"audit.configuration", 
"tenant": 
"tenantid", 
"customDetails": 
"": 
"]}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID</td>
<td>Update Application ID</td>
<td>• action: update</td>
<td>objectType: APPLICATION</td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
"message": "{"uid": "<msg GUID>",
"user": "<user email>",
"time": "2021-07-13T05:58:25.864Z",
"id": "<msg GUID>",
"object": {
"type": "user",
"id": {
"class": "com.sap.it.spc.apimgmt.entityhandlers.Do
mainHook",
"objectId": "Application",
"action": "aboutToUpdate",
"message": "application.
update",
"user": "<user email>",
"attributes"
}
[("Application
Id"],
"value"]},
"status": "BEGIN",
"category": "audit.config
uration",
"tenant": "tenantid",
"customDetails": {}}
```
## Event grouping

<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID</td>
<td>Delete Application ID</td>
<td>- action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- objectType: APPLICATION</td>
<td></td>
</tr>
</tbody>
</table>

### Audit Log

Example:

```
{
  "message": {
    "<msg GUID>": 
    "<user email>": 
    "time": 
    "2021-07-13T05:58:25.864Z
    "$oid": 
    "<msg GUID> 
    "object": 
    { "type": "<msg GUID> 
    "user": 
    { "class": 
    "com.sap.it.spc.apimgmt.entityhandlers.Do 
    mainHook": 
    "objectId": 
    "Application 
    "action": 
    "aboutToDelete 
    "message": 
    "application. 
    delete", 
    "user": 
    "<user email>")
    "attributes 
    "Application Id": 
    "value": 
    "status": 
    "BEGIN", 
    "category": 
    "audit.configuration", 
    "tenant": 
    "tenantid", 
    "customDetails 
    :{}"
  
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscription</td>
<td>Create Subscription</td>
<td>• action: create</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: SUBSCRIPTION</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
{ "message": "\"<msg GUID>\", "user": "<user email>\", "time": "2021-07-13T05:58:25.864Z\", "id": "<msg GUID>\", "object": {"type": "user","id":{"class": "com.sap.it.spc.apimgmt.entityhandlers.Do meinHook"}, "objectId": "Subscription","action": "aboutToCreate","message": "subscription.create","user": "<user email>"}, "attributes ": [{"Application Id":"","value":[]}], "status": "BEGIN", "category": "audit.configuration", "tenant": "tenantid", "customDetails": {}}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscription</td>
<td>Update Subscription</td>
<td>• action: update</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: SUBSCRIPTION</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
"message": {
  "uid": "<msg GUID>",
  "user": "<user email>",
  "time": "2021-07-13T05:58:25.864Z",
  "id": "<msg GUID>",
  "object": {
    "type": "SUBSCRIPTION",
    "objectId": "Subscription",
    "action": "aboutToUpdate",
    "message": "application.update",
    "user": "<user email>",
    "attributes": {
      "Application Id": "",
      "isSubscription": "false",
      "status": "BEGIN",
      "category": "audit.configuration",
      "tenant": "tenantid",
      "customDetails": {}
    }
  }
}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| Key Map Entry        | Create Key Map Entry        | - action: create  
- objectType: KEY MAP ENTRY                                                                 | Key Value Map [page 461] |

Audit Log

Example:

```json
"\"uid\": "<msg GUID>\n","\"user\": "<user email>"\n","\"time\": "2021-06-09T10:34:32.369Z"\n","\"id\": "<ID>\n","\"success\n":true,"\"object\": {
"type": "KEY MAP ENTRY"}
,"\"id\": {
"class": "com.sap.apimgmt.asmprov.core.processor.KeyMapEntryProcessor","\"action\": "\"create\", 
"message": "<based on the event the corresponding message will be logged>"}"
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Map Entry</td>
<td>Update Key Map Entry</td>
<td>• action: update</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: KEY MAP ENTRY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
"{"uuid": "<msg GUID>", "msg": "<user e-mail>", "time": "2021-06-09T00:34:32.369Z", "id": "<ID>", "success": true, "object": {"type": "KEY MAP ENTRY"}, "id": {"class": "com.sap.apim.gmt.asmprov.core.processor.KeyMapEntryProcessor"}, "objectID": "<KeyMapEntry>", "action": "update", "message": "<based on the event the corresponding message will be logged>"}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Map Entry</td>
<td>Delete Key Map Entry</td>
<td>• action: delete&lt;br&gt;• objectType: KEY MAP ENTRY</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
{ "uuid": "<msg GUID>",
  "<msg GUID",
  "<user email>":
  "<user e-mail>",
  "time": "2021-06-09T00:34:32.369Z",
  "id": "<ID>",
  "success": true,
  "object": {
    "type": "KEY MAP ENTRY",
    "id": {
      "class": "com.sap.apimgmt.asmprov.core.processor.KeyMapEntryProcessor",
      "objectId": "KeyMapEntry",
      "action": "delete",
      "message": "<based on the event the corresponding message will be logged>"
  }
```

<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Map Value</td>
<td>Create Key Map Value</td>
<td>• action: create</td>
<td>Create a Key Value Map [page 462]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: KEY MAP VALUE</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
"\"uuid\":\"<msg GUID>\",\"\"user\":\"<user e-mail>\",\"time\":\"2021-06-09T10:34:32.369Z\",\"id\":\"<ID>\",\"success\":true,\"object\":\{"type":"KEY MAP VALUE\",\"id\":\{"class":\"com.sap.apimgmt.asmprov.core.processor.KeyMapValueProcessor\",\"action":"create",\"message":"<based on the event the corresponding message will be logged>"\n```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Map Value</td>
<td>Update Key Map Value</td>
<td>• action: update</td>
<td>Update a Key Value Map</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: KEY MAP VALUE</td>
<td>[page 463]</td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
"{\
"uuid": \\
"<msg GUID>",
"","user": \\
"<user e- \\
"mail>",
"time": \\
"2021-06-09T1 \\
0:34:32.369Z \\
", \\
"id": \\
"<ID> \\
",
"success \\
":"true,
"object": \\
{"type": \\
"KEY MAP \\
VALUE"},
"id": \\
{"class": \\
"com.sap.apim \\
gmt.asmprov.co \\
re.processor.K \\
eyMapValueProc \\
essor"},
"objectId": \\
"KeyMapEntry \\
", \\
"action": \\
"update", \\
"message": \\
"<based on the event the corresponding message will be logged>"
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Map Value</td>
<td>Delete Key Map Value</td>
<td>• action: delete</td>
<td>Delete a Key Value Map [page 464]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: KEY MAP VALUE</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
{ "uuid": "<msg GUID>",
 "<msg GUID>",
 "<user e-mail>":
 "<user e-mail>",
 "time": "2021-06-09T10:34:32.369Z",
 "id": "<ID>",
 "success": "true",
 "object": {
 "type": "KEY MAP VALUE",
 "id": {
 "class": "com.sap.apimgmt.asmprov.core.processor.KeyMapValueProcessor",
 "objectId": "KeyMapEntry",
 "action": "delete",
 "delete": "",
 "message": "<based on the event the corresponding message will be logged>"}}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Store</td>
<td>Create Certificate Store</td>
<td>• action: create</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: CERTIFICATE STORE</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
"\"uid\":\"
"\"msg GUID\":\"
"\"user\":\"
"\"user e-mail\":\"
"\"time\":\"2021-06-09T00:34:32.369Z"
"\"
"\"id\":\"<ID>
"\"
"\"success\":\"true,
"\"object\":\{"\"type\":\"CERTIFICATE STORE\",
"\"id\":\"<ID>
"\"
"\"class\":\"com.sap.api mgmt.asmprov.core.processor.CertificateStoreProcessor\",
"\"action\":\"create",
"\"message\":\"<based on the event the corresponding message will be logged>\"```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Store</td>
<td>Update Certificate Store</td>
<td>• action: update</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: CERTIFICATE STORE</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
{
"uuid": "<msg GUID>",
"<msg GUID>":
"user":
"<user e-mail>",
"time":
"2021-06-09T10:34:32.369Z",
"id": "<ID>",
"success":true,
"object":
{"type":
"CERTIFICATE STORE"},
"id":
{"class":
"com.sap.apimgmt.asmprov.core.processor.CertificateStore"},
"objectID":
"CertificateStore",
"action":
"update",
"message":
"<based on the event the corresponding message will be logged>"
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Store</td>
<td>Delete Certificate Store</td>
<td>• action: delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: CERTIFICATE STORE</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
"{"uuid": "<msg GUID>",
"<msg GUID>",
"<user":
"<user email>",
"time":
"2021-06-09T10:34:32.369Z",
"id": "<ID>",
"success":true,
"object":
{"type":
"CERTIFICATE STORE"},
"id":
{"class":
"com.sap.apimgmt.asmprov.core.processor.CertificateStoreProcessor"},
"objectID":
"CertificateStore"},
"action":
"delete",
"message":
"<based on the event the corresponding message will be logged>"
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| Certificate    | Create Certificate     | • action: create  
• objectType: CERTIFICATE | Manage Certificates [page 371] |

Audit Log

Example:

```json
{"uuid": 
"<msg GUID>",
"user": 
"<user email>",
"time": 
"2021-06-09T10:34:32.369Z",
"id": 
"<ID>",
"success": 
"true",
"object": {
"type": 
"CERTIFICATE",
"id": {
"class": 
"action": 
"create",
"message": 
"<based on the corresponding message will be logged>"}}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate</td>
<td>Update Certificate</td>
<td>• action: update</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• objectType: CERTIFICATE</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log

Example:

```
{"uuid":"<msg GUID>","user":"<user email>","time":"2021-06-09T10:34:32.369","id":"<ID>","success":true,"object":{"type":"CERTIFICATE","id":{"class":"com.sap.bpmm.bpmm.core.processor.CertificateProcessor","objectId":null,"Certificate":null,"action":"update","message":"<based on the event the corresponding message will be logged>"}}
```
<table>
<thead>
<tr>
<th>Event grouping</th>
<th>What events are logged</th>
<th>How to identify related log events</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate</td>
<td>Delete Certificate</td>
<td>- action: delete</td>
<td>- objectType: CERTIFICATE</td>
</tr>
</tbody>
</table>

Audit Log

Example:

```json
{"uuid": "<msg GUID>",
"<msg GUID>",
"<user": "<user e-mail>",
"<user e-mail>",
"<time": "2021-06-09T10:34:32.369Z",
"<time": "2021-06-09T10:34:32.369Z",
"<id": "<ID>",
"<ID>",
"success": true,
"success": true,
"object": {
"object": {
"type": "CERTIFICATE",
"type": "CERTIFICATE",
"objectId": "Certificate",
"objectId": "Certificate",
"id": {
"id": {
"class": "com.sap.apim gmt.asmprov.co re.processor.CertificateProcessor",
"class": "com.sap.apim gmt.asmprov.co re.processor.CertificateProcessor",
"action": "delete",
"action": "delete",
"message": "$based on the event the corresponding meassage will be logged}"
"message": "$based on the event the corresponding meassage will be logged}"
```

Related Information

Audit Logging in the Cloud Foundry Environment
Audit Logging in the Neo Environment
1.8 Monitoring and Troubleshooting

Information on creating tickets for reporting incidents and errors.

Reporting an Incident

You can report an incident or error through the SAP Support Portal. For more information, see Product Support.

Please use the following component for your incident:

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPU-API-OD</td>
<td>SAP API Management - On Demand</td>
</tr>
</tbody>
</table>

When submitting the incident we recommend including the following information:

- Landscape information (Canary, EU10, US10)
- The URL of the page where the incident or error occurs
- The steps or clicks used to replicate the error
- Screen grabs, videos, or the code being inputted

Frequently Asked Questions

For the list of answers to common questions about SAP API Management, see Frequently Asked Questions.

1.8.1 Limits

This topic describes the product configuration and the naming conventions for API Management.

Consider the boundary conditions mentioned in the following tables for building, managing, and reviewing the APIs. API Management is designed to perform at its maximum, when it is configured within the specified conditions. Exceeding these values leads to:

- High API latency
- Low API throughput
- Failing API calls.

Currently, certain configurations are automatically enforced for optimal performance.

Product Configurations
<table>
<thead>
<tr>
<th>Feature</th>
<th>Specified Configuration Values</th>
<th>Automatically Enforced</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API Proxies</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API proxy resource file size (such as XSL, JavaScript or Python).</td>
<td>15 MB</td>
<td>Yes</td>
</tr>
<tr>
<td>Maximum number of resources that can be attached to an API proxy</td>
<td>By default, up to 100 resources can be attached to an API proxy. It is recommended that more than 100 resources should not be attached as it might lead to a timeout while updating or deploying an API proxy. In case you have a business requirement to attach more than 100 resources to an API proxy, please contact the SAP API Management support team by creating a ticket with component OPU-API-OD-DT. However, the team can support your request up to a maximum of 200 resources per API proxy.</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Cache and KVM</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caches</td>
<td>100 MB</td>
<td>No</td>
</tr>
<tr>
<td>Cache key size</td>
<td>2 KB</td>
<td>Yes</td>
</tr>
<tr>
<td>Cache value size</td>
<td>512 KB</td>
<td>Yes</td>
</tr>
<tr>
<td>Cache expiration</td>
<td>&gt;=180 seconds, &lt;=30 days</td>
<td>No</td>
</tr>
<tr>
<td>Key Value Map (KVM) key size</td>
<td>2 KB</td>
<td>Yes</td>
</tr>
<tr>
<td>Key Value Map (KVM) value size</td>
<td>10 KB</td>
<td>No</td>
</tr>
<tr>
<td><strong>Keys, Developers, Apps, Products</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API key size</td>
<td>2 KB</td>
<td>Yes</td>
</tr>
<tr>
<td>Custom attribute name size</td>
<td>255 characters</td>
<td>Yes</td>
</tr>
<tr>
<td>Custom attribute value size</td>
<td>1024 characters</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of custom attributes permitted</td>
<td>18</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>OAuth</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OAuth access token expiration</td>
<td>&gt;=180 seconds, &lt;=30 days</td>
<td>No</td>
</tr>
<tr>
<td>OAuth refresh token expiration</td>
<td>&gt;=1 day, &lt;=90 days</td>
<td>No</td>
</tr>
<tr>
<td>OAuth access and refresh token size</td>
<td>2 KB</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>System</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API proxy request URL size</td>
<td>7 KB</td>
<td>Yes</td>
</tr>
<tr>
<td>Request header size</td>
<td>30 KB</td>
<td>Yes</td>
</tr>
<tr>
<td>Response header size</td>
<td>25 KB</td>
<td>Yes</td>
</tr>
<tr>
<td>Feature</td>
<td>Specified Configuration Values</td>
<td>Automatically Enforced</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Request size (for non-streamed HTTP requests)</td>
<td>10 MB</td>
<td>Yes</td>
</tr>
<tr>
<td>Request size (for streamed HTTP requests)</td>
<td>&lt;500 MB. However, if the connection is terminated unexpectedly, you must re-initiate the connection.</td>
<td>No</td>
</tr>
<tr>
<td>Response size (for non-streamed HTTP requests)</td>
<td>10 MB</td>
<td>Yes</td>
</tr>
<tr>
<td>Timeout (Applicable for all API Proxies and the Backend Server is expected to respond within the value set)</td>
<td>55 Seconds</td>
<td>Yes</td>
</tr>
<tr>
<td>Security Protocol</td>
<td>TLSv 1.2 only (on Hyperscalers)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>TLSv 1.2 &amp; TLSv1.1 (on SAP DC)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>TLSv1.1 (deprecated, planned to be removed by end of 2019)</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>TLSv1.0 (Unsupported)</td>
<td>Yes</td>
</tr>
<tr>
<td>SNI (Server Name Indication) Enforcement</td>
<td>In API Management, the client is expected to pass a SNI extension (server_name) with target endpoint hostname as part of the initial TLS handshake. Based on the server_name SNI extension the APIM servers will determine the certificate that would be served to the client. For Client which doesn’t support SNI extension, APIM would send a default certificate which is provided by SAP. For Reference on SNI: <a href="https://en.wikipedia.org/wiki/Server_Name_Indication">https://en.wikipedia.org/wiki/Server_Name_Indication</a></td>
<td>Yes</td>
</tr>
<tr>
<td>Security Protocol applicable for API Management Runtime</td>
<td>TLSv 1.2 only (on Hyperscalers &amp; SAP DC)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>TLSv1.0 &amp; TLS 1.1 (Unsupported)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Naming Conventions

Table below describes the naming constraints for API Management.

<table>
<thead>
<tr>
<th>Name</th>
<th>Maximum Characters</th>
<th>Permitted Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>API product</td>
<td></td>
<td>Alphanumeric, space, and the following: _ - . # $ %</td>
</tr>
</tbody>
</table>
### 1.8.2 Monitor the Health of Custom Domain Virtual Host Certificates Using SAP Cloud ALM

You can use SAP Cloud Application Lifecycle Management (ALM) application to proactively detect issues and monitor the health of custom domain virtual host certificates in API Management.

The SAP Cloud Application Lifecycle Management (ALM) platform allows you to monitor environment backlogs and status of automation processes regarding execution status, application status, start delay, and runtime of various SAP Cloud solutions and services. To integrate the Cloud Application Lifecycle Management (ALM) application with the Health service endpoint, refer to [SAP Cloud ALM Onboarding](#).

The API `https://<host>:<port>/api/1.0/Health`, which has been provisioned in API Management, can be integrated with Cloud Application Lifecycle Management (ALM) to provide information about the custom domain virtual host certificates expiry details. You can use this API to read information about the certificates provided by the customers for their custom domain virtual hosts.

**Note**

You have to enable API access with `APIPortal. Administrator` role to use the `https://<host>:<port>/api/1.0/Health` API. To access this API from the Cloud Application Lifecycle Management (ALM) application, use `OAuth` authentication.
2 Migration Assistance for API Management from Neo to Cloud Foundry Environment

Choose to migrate an existing API Management subscription that you have in the Neo environment to another API Management subscription in the public cloud infrastructures (hyperscalers) within the Cloud Foundry environment.

Migration Assistant for asset migration includes the tools and utilities that enable migration of design time assets nondisruptively from the Neo to the Cloud Foundry environment.

Your source system is the system that has your API Management subscription in the Neo environment.

Your target system is the system that has your API Management subscription on the hyperscalers-managed infrastructure within the Cloud Foundry environment. Here Cloud Foundry environment can be your native standalone API Management subscription or API Management capability within the Integration Suite.

For the migration assistance, you must have an API Management subscription in a subaccount within the Cloud Foundry environment.

After completing the prerequisites mentioned in the steps below, you can clone your API Management artifacts nondisruptively from the source to the target system. Post cloning, you must complete some user actions and validate your target system.

**i Note**

The Developer portal is renamed to API business hub enterprise in Cloud Foundry environment. In this document API business hub enterprise is referred to as Developer portal even in Cloud Foundry environment.

The steps assisting the migration of your API Management from your source system to a target system are:

1. Prerequisites [page 624]
2. Clone API Management Artifacts [page 626]
3. Post Cloning Tasks [page 644]

2.1 Prerequisites

Checks to be completed before you start migrating your API Management artifacts nondisruptively from your source system to a target system.

Your source system is the system that has your API Management subscription in the Neo environment.

Your target system is the system that has your API Management subscription on the hyperscalers-managed infrastructure within the Cloud Foundry environment.
Prerequisites for the source system

- You must have a valid API Management system (API portal and Developer Portal) running in the Neo environment.
- The source system must support basic authentication for API access on API portal and Developer Portal.
- Note the API portal and Developer Portal URLs of the source system and keep handy.
- You must have identified a user with the following roles assigned in your source API portal and Developer Portal systems:
  - APIPortal.Administrator role in the API portal
  - AuthGroup.API.Admin role in the Developer Portal

Keep the credentials of this user handy. These credentials are used while filling in the details of the `apim-tct-input.json` file before running the Tenant Cloning Tool. See Clone API Management Artifacts [page 626].

Prerequisites for the target system

- If API Management is not already enabled on your target system, complete the set-up. See Initial Setup [page 42].
  Check whether the API Management service broker service instance is created with the Starter Plan in the same subaccount.

<table>
<thead>
<tr>
<th>Service Instance for Starter Plan in the Subaccount</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Already present</td>
<td>You cannot use this subaccount for migration. Create a new subaccount in the hyperscalers-managed infrastructure within the cloud foundry environment and enable API Management on that subaccount for it to act as your target system. Additionally, if you want to reuse the existing runtime then follow the steps mentioned in the Migrating API Management Subscription Created Using the Starter Plan Service Instance [page 650].</td>
</tr>
<tr>
<td>Not present</td>
<td>You can choose to reuse this account as your target system for migration, or create a new subaccount.</td>
</tr>
</tbody>
</table>

If you have already enabled API Management on your target system, and want to reuse the same for migration:
- It’s recommended that you do not have any pre-existing entities such as API proxies or products on this system.

**Note**

Any entity, if pre-existing in your target API Management subscription, can be over-written during the cloning process.

- If your target system is connected to a custom IDP, ensure that your IDP is configured correctly and mapping for the details like your first name, last name, email ID, and user ID is done.
- Ensure that API access is enabled for the API portal and the Developer Portal for the following roles:
  - APIPortal.Administrator role in the API portal
  - AuthGroup.API/Admin role in the Developer Portal
Note the service keys (url, tokenurl, clientId, and clientSecret) for the given roles, and keep handy.

To know more about API access plan for API portal, see API Access plan for API Portal [page 56].

To know more about API access plan for Developer Portal, see API Access Plan for API business hub enterprise [page 59].

Follow the steps in API Access Plan for API business hub enterprise [page 59], without which the cloning of the Developer portal entities might fail.

- When you have API Products protected by the custom roles permission in the source Neo system, ensure that custom roles creation and assignments are done in the target Cloud Foundry environment before starting the migration.

Once you complete these checks, you can start cloning your API Management artifacts from the source to the target system. See Clone API Management Artifacts [page 626].

2.2 Clone API Management Artifacts

Clone the API Management artifacts using the Tenant Cloning tool.

Once you have your source and target system ready, you can clone your API Management artifacts to the target system by running the Tenant Cloning Tool that you downloaded from here.

Prerequisites

- You must have downloaded the Tenant Cloning Tool () from the link provided above.APIM-TCT-<version>.zip
- APIM-TCT-You must have extracted the contents of the APIM-TCT-<version>.zip file into a folder (example name apim-tct).

  This extracted folder must contain:
  - a java apim-tct-client-<version>.jar file
  - a sample apim-tct-input.json file
  - a lib folder (this folder and its contents must not be modified)
  - a README.md file
  - Script files to download open-source libraries that are required to run the apim-tct-client-<version>.jar file:
    - download_dependencies.ps1 for Windows systems
    - download_dependencies.sh for Mac and Linux systems

  i Note
  Download the dependencies as described in the Downloading the Dependencies section.
Note
If you are using the version of the Tenant Cloning Tool prior to 1.5.2, make sure that you update to the latest version 1.5.2 or above. This is done to handle the critical vulnerability CVE-2021-44228 and CVE-2021-45046, which was detected in the open-source library log4j2.

- The system running the API Management Tenant Cloning Tool must have Java Runtime Environment 8 or above supported.
- Microsoft Excel File Reader

Downloading the Dependencies

For Windows Systems:
- Open the PowerShell terminal.
- Go to the apim-tct folder in the terminal.
- Run the .\download_dependencies.ps1 command. The required libraries are downloaded to the lib folder.

For Mac and Linux Systems:
- Open the default terminal from your system.
- Go to the apim-tct folder in the terminal.
- Run the chmod +x download_dependencies.sh command to make the file executable.
- Run the .\download_dependencies.sh command. The required libraries are downloaded to the lib folder.

Note
If you encounter an error while running these commands, then you can download the dependencies manually from the link provided in the script file and place them into the lib folder.

Procedure

1. Fill in the apim-tct-input.json file by providing details such as the URLs of your source and target systems, access token URLs, and usernames and passwords to your target systems. Ensure that you don’t modify the name of the apim-tct-input.json file.
Structure of the apim-tct-input.json file:

<table>
<thead>
<tr>
<th>Input Field</th>
<th>Data Type</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>apiportal</td>
<td>url</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Required</td>
<td>URL of the source API management, API portal in the Neo environment Example: https://&lt;application_name&gt;&lt;provider_subaccount&gt;-&lt;consumer_subaccount&gt;.&lt;domain&gt;</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
<td>Optional</td>
<td>User ID having the APIPortal.Administrator role in the above subscription You’re prompted to enter these values while running the command in Step 3 if you have not already provided these details in the apim-tct-input.json file.</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>Optional</td>
<td>Password of the above user You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the apim-tct-input.json file.</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>devportal</td>
<td>url</td>
<td>Required</td>
<td>URL of the source API Management, Developer Portal in the Neo environment. Example: https://&lt;application_name&gt;&lt;provider_subaccount&gt;&lt;consumer_subaccount&gt;.&lt;domain&gt;</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
<td>Optional</td>
<td>User ID having the AuthGroup.AP I.Admin role in the above subscription. You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the apim-tct-input.json file.</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>Optional</td>
<td>Password of the above user. You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the apim-tct-input.json file.</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>cfSubaccountTenantID</td>
<td>String</td>
<td>Optional</td>
<td>This is the Tenant ID for your Cloud Foundry subaccount where starter plan service instance is enabled.</td>
</tr>
</tbody>
</table>

**Note**
If you are migrating within the same subaccount, you are not required to add this parameter.

This parameter is mandatory if you are migrating to a Cloud Foundry subaccount, which is different from your existing starter plan subaccount.

**Note**
Navigate to the cockpit to fetch the Cloud Foundry Tenant ID for the subaccount where the starter plan service instance ex-
<table>
<thead>
<tr>
<th>Input Field</th>
<th>Data Type</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>apiportal</td>
<td><strong>Url</strong></td>
<td>String</td>
</tr>
<tr>
<td>tokenUrl</td>
<td>String</td>
<td>Required</td>
<td>Token URL received during creation of the service key for API portal API access for the <code>APIPortal.Administrator</code> role. To know more about creating the service key, see API Access plan for API Portal [page 56].</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>clientId</td>
<td>String</td>
<td>Optional</td>
<td>The client ID received during creation of the service key for API portal API access for the APIPortal.Administrator role. You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the apim-tct-input.json file. To know more about creating the service key, see API Access plan for API Portal [page 56].</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>clientSecret</td>
<td>String</td>
<td>Optional</td>
<td>The client secret received during creation of the service key for API portal API access for the <strong>APIPortal.Administrator</strong> role. You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the <code>apim-tct-input.json</code> file. To know more about creating the service key, see API Access plan for API Portal [page 56].</td>
</tr>
<tr>
<td>apiportalSelfServiceAdmin.Url</td>
<td>String</td>
<td>Required</td>
<td>URL received during creation of the service key for API portal API access for the <strong>APIManagement.SelfService.Administrator</strong> role. To know more about creating the service key, see API Access plan for API Portal [page 56].</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>tokenUrl</code></td>
<td>String</td>
<td>Required</td>
<td>Token URL received during creation of the service key for API portal API access for the <code>APIManagement.SelfService.Administrator</code> role. To know more about creating the service key, see API Access plan for API Portal [page 56].</td>
</tr>
<tr>
<td><code>clientId</code></td>
<td>String</td>
<td>Optional</td>
<td>The client ID received during creation of the service key for API portal API access for the <code>APIManagement.SelfService.Administrator</code> role. You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the <code>apim-tct-input.json</code> file. To know more about creating the service key, see API Access plan for API Portal [page 56].</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>clientSecret</td>
<td>String</td>
<td>Optional</td>
<td>The client secret received during creation of the service key for API portal API access for the API Management SelfService.Administrator role. You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the <code>apimtct-input.json</code> file. To know more about creating the service key, see API Access plan for API Portal [page 56].</td>
</tr>
<tr>
<td>devportal url</td>
<td>String</td>
<td>Required</td>
<td>URL received during creation of the service key for Developer Portal API access for the AuthGroup.API.Admin role. See API Access Plan for API business hub enterprise [page 59].</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>tokenUrl</strong></td>
<td>String</td>
<td>Required</td>
<td>Token url received during creation of the service key for API business hub enterprise API access for the <strong>AuthGroup.AP I.Admin</strong> role. To know more about creating the service key, see API Access Plan for API business hub enterprise [page 59].</td>
</tr>
<tr>
<td><strong>clientId</strong></td>
<td>String</td>
<td>Optional</td>
<td>The client ID received during creation of the service key for Developer Portal API access for the <strong>AuthGroup.AP I.Admin</strong> role. You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the apim-tct-input.json file. To know more about creating the service key, see API Access Plan for API business hub enterprise [page 59].</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>clientSecret</strong></td>
<td>String</td>
<td>Optional</td>
<td>The client secret received during creation of the service key for Developer Portal API access for the <strong>AuthGroup.AP I.Admin</strong> role. You’re prompted to enter these values while running the command in Step 3 if you haven’t already provided these details in the <code>apim-tct-input.json</code> file. To know more about creating the service key, see API Access Plan for API business hub enterprise [page 59].</td>
</tr>
<tr>
<td>Input Field</td>
<td>Data Type</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>skipApplicationKeySecretCloning</td>
<td>Supported values: Boolean</td>
<td>Optional</td>
<td>○ The default value for skipApplicationKeySecretCloning is false. Note: If you want to skip the cloning of Application Key and Secret in side by side migration, then set the “skipApplicationKeySecretCloning” flag to true.</td>
</tr>
<tr>
<td>clone</td>
<td>Supported values: Boolean</td>
<td>Optional</td>
<td>○ The default value for skip-apiportal is false, and API portal entities are cloned. ○ If you set the value for skip-apiportal to true, no cloning of the API portal entities takes place.</td>
</tr>
</tbody>
</table>

**SAP API Management**

Migration Assistance for API Management from Neo to Cloud Foundry Environment
<table>
<thead>
<tr>
<th>Input Field</th>
<th>Data Type</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
</table>
| skip-devportal | Boolean       | Optional          | ○ The default value for skip-devportal is false, and Developer Portal entities are cloned.  
            | true/false     |                   | ○ If you set the value for skip-devportal to true, no cloning of the Developer Portal entities takes place.  
            |                |                   | ○ If skip-apiportal is set to false, but skip-devportal to true, then only API portal entities are cloned.  
            |                |                   | ○ If skip-apiportal is set to true, but skip-devportal to false, then only Developer Portal entities are cloned and cloning for entities (like applications) may fail, pertaining to nonavailability of dependent entity (like API Product) in Developer Portal. |

**stage**

Supported values: string

- "DEFAULT"  
- "SWITCHOVER"

Optional

The supported values for this parameter is either default or switchover.

*** apiportalSelfServiceAdmin This input field is mandatory for Starter Plan migration.

*** API portal credentials for source and target for all scenarios are mandatory.

→ Remember

For the clone input attribute:

○ Both skip-apiportal and skip-devportal are set to false by default, so, API portal entities are cloned first, followed by Developer Portal entities.

○ If both skip-apiportal and skip-devportal are set to true, no cloning takes place.

○ If skip-apiportal is set to false, but skip-devportal is set to true, then only the API portal entities are cloned.

○ If skip-apiportal is set to true, but skip-devportal to false, then only Developer Portal entities are cloned and cloning for entities (like applications) may fail, pertaining to nonavailability of dependent entity (like API Product) in Developer Portal.

Sample configuration:

```json
{
    "source": {
        "apiportal": {
            "url": "<URL of Source (Neo based) API Portal>",
            "username": "<user id having APIPortal.Administrator role in above subscription>",
            "password": "<password of the above user>"
        },
        "devportal": {
            "url": "<URL of Source (Neo based) Developer Portal>",
            "username": "<user id having AuthGroup.API.Admin role in above subscription>",
            "password": "<password of the above user>"
        }
    }
}```
2. Run the following commands from your Java command-line interface to verify the setup and check the version of the tool. This is an optional step.

- To verify the setup:
  ```
  java -jar apim-tct-client-<version>.jar verify
  ```
- To check the version of the tenant cloning tool you’re using:
  ```
  java -jar apim-tct-client-<version>.jar version
  ```

3. To begin the cloning process, run the following command from your Java command-line interface:

   ```
   java -jar apim-tct-client-<version>.jar
   ```

Result

Your API Management entities are now cloned to your target system.
An excel file named apimtct-output.xlsx and a log file named apimtct-logs.log are generated in the same folder where the .jar file is present.

The status of each cloned entity is stored in a separate worksheet within the output excel file.
Structure of a Worksheet Within apimtct-output.xlsx File

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Entity ID</td>
</tr>
<tr>
<td>Name</td>
<td>Entity name</td>
</tr>
<tr>
<td>Type</td>
<td>Entity type</td>
</tr>
<tr>
<td>Script Execution Timestamp (UTC)</td>
<td>Script execution time in UTC</td>
</tr>
<tr>
<td>Artifact's Last Modified Timestamp (UTC)</td>
<td>Last modified time of the entity in the source API Management system (UTC)</td>
</tr>
<tr>
<td>STATUS</td>
<td>Migration Status:</td>
</tr>
<tr>
<td></td>
<td>○ SUCCESS (Entity successfully cloned)</td>
</tr>
<tr>
<td></td>
<td>○ FAILURE (Entity failed to clone)</td>
</tr>
<tr>
<td></td>
<td>○ SKIPPED (Cloning of Entity skipped)</td>
</tr>
</tbody>
</table>

You can view the status of the cloned artifacts in the apimtct-output.xlsx file or in the apimtct-logs.log file.

**Note**
- Ensure that the apimtct-output.xlsx file isn’t open while you run the script.
- It’s recommended that you don’t modify the apimtct-output.xlsx file.

**Troubleshooting During Cloning:**
- If the Tenant Cloning Tool shuts down unexpectedly, restart and try again.
  - If the tool throws an error repeatedly while running, you can report the incident or error on the component OPU-API-OD-DT through the SAP Support Portal.

**Next Steps**

After the cloning process completes, you must perform the tasks mentioned in the User Actions worksheet within the output excel file apimtct-output.xlsx.

To know more about what actions you must take, see the User Actions section in Post Cloning Tasks [page 644].

To know more about the entities that are cloned and the entities that aren’t cloned, see Cloned and Uncloned Entities [page 642].
2.2.1 Cloned and Uncloned Entities

Refer this section for the entities that are cloned and entities that aren't cloned during the migration process.

Entities That Are Cloned

- **Note**
  - Currently, when a custom role is assigned to a Product, the Application creation using the tenant cloning tool is not supported.
  - As a work-around, before initiating the cloning process, remove the custom role assigned to the Product in the Source system and proceed with the cloning process.
  - After the cloning process is completed, reassign the custom roles to the Product in the Source system. Also, ensure that the custom roles are assigned to the Product in the Target system.
  - In case the custom roles aren't appearing in the Permission tab, as mentioned in the Prerequisite section, ensure that the custom roles are created and assigned to the developers in the target Cloud Foundry environment.

The following list displays the API Management entities that are cloned:

- Certificates and Certificate Store
- Rate Plans
- Key Value Maps
- API Providers
- Policy Templates
- API Proxies
- API Products
- Measure Codes for Custom Measures
- Dimension Codes for Custom Dimensions
- Application
- Application Developer
- Access Control Permissions for API Product
- Custom Metrics and Charts

Entities That Are Not Cloned

The following list displays the API Management entities that aren’t cloned, including sensitive data like your certificates and credentials.

- **Sensitive Data**
  - Certificates
  - Encrypted Key Value Maps
2.2.2 Tenant Cloning Tool Behavior

This topic describes the behavior of the Tenant Cloning Tool with respect to cloning some of the entities from your source system.

- If you add or modify an entity in your source system, it is always cloned to the target system in your subsequent run of the Tenant Cloning Tool.
- If you add a new entity to your target system at any point, it is retained in the target system after the subsequent run of the Tenant Cloning Tool, irrespective of whether the entity is present in your source system or not.
- Newer state of an existing entity present in your source system is always migrated to the target system after the subsequent run of the Tenant Cloning Tool, and overwrites any older state of the entity in the target.
- During the cloning of the Developer Portal entity Application Developer, the app developer receives email notifications while being onboarded to the target Developer Portal. We recommend that you inform your developers about the impending migration and email notifications that they might receive during the process.
- Custom Charts are cloned to the target API portal as many times as you run the Tenant Cloning Tool.
- All the API Proxies are cloned onto the default virtual host.
- Post cloning, the API proxies on the target system are in active and deployed state. You must reapply the desired states to the proxies. To know more about API proxy states, see API Proxy States [page 393].
- Cloning of Custom chart is now supported for migrating API Management subscription created using the Starter Plan service instance.
2.3 Post Cloning Tasks

Post the completion of the cloning process, you must perform some actions, checks, and validations. The following sections explain the tasks that you must perform after the cloning of your API Management artifacts from the Neo to the Cloud Foundry environment is complete.

User Actions

You can view the status of the cloned artifacts in the `apim-tct-output.xlsx` excel file or in the `apim-tct.log` file, generated in the same folder where the .jar file is present.

Perform the tasks mentioned in the User Actions worksheet within the `apim-tct-output.xlsx` excel file.

The following table describes the actions required for each cloned entity:

<table>
<thead>
<tr>
<th>Cloned Entity</th>
<th>User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificates</td>
<td>All the certificates that are cloned to the target system are dummy certificates.</td>
</tr>
<tr>
<td></td>
<td>Perform the following steps:</td>
</tr>
<tr>
<td></td>
<td>1. From your source system, note down the certificate names and the corresponding certificate store name.</td>
</tr>
<tr>
<td></td>
<td>2. From your target system, delete the dummy certificates that were cloned:</td>
</tr>
<tr>
<td></td>
<td>1. In your target API Management, API portal, navigate to <code>Configure Certificates</code>.</td>
</tr>
<tr>
<td></td>
<td>2. Select the cloned dummy certificate that you want to delete.</td>
</tr>
<tr>
<td></td>
<td>3. Click the delete icon under the Actions column.</td>
</tr>
<tr>
<td></td>
<td>3. In your target API Management, API portal, upload the relevant certificates, providing the same names and under same certificate store as present in your source system.</td>
</tr>
<tr>
<td></td>
<td>1. In your target API Management, API portal, navigate to <code>Configure Certificates</code>.</td>
</tr>
<tr>
<td></td>
<td>2. Click Create.</td>
</tr>
<tr>
<td></td>
<td>3. In the Create Certificate window, provide the details and upload the certificate.</td>
</tr>
</tbody>
</table>
### Cloned Entity

<table>
<thead>
<tr>
<th>Key Value Maps</th>
<th>User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill in the values for the keys of the encrypted Key Value Maps.</td>
<td></td>
</tr>
<tr>
<td>1. In your target API Management, API portal, navigate to Configure Key Value Maps.</td>
<td></td>
</tr>
<tr>
<td>2. Click on the encrypted key value map.</td>
<td></td>
</tr>
<tr>
<td>3. In the Edit Key Value Map window, provide the details and click Save.</td>
<td></td>
</tr>
</tbody>
</table>

### API Provider Credentials

Provide the Basic Auth password (if present in your source system) for an API Provider.

1. In your target API Management, API portal, navigate to Configure API Providers. |
2. Click on the desired API provider. |
3. In the View API Provider window, click Catalog Service Settings Edit. |
4. Provide the basic auth password for the API provider and click Save. |

**Update open connector credentials:**

1. In your target API Management, API portal, navigate to Configure API Providers. |
2. Click on the desired API provider. |
3. In the View API Provider window, click Catalog Service Settings Edit. |
4. Provide the Org ID and User Secret from your corresponding Open Connector subscription.

### Proxy Scoped Key Value Map

Provide instance token value in the proxy scoped Key Value Map.

1. In your target API Management, API portal, navigate to Develop APIs. |
2. Click on the desired API proxy. |
3. In the View API page, scroll down to Key Value Map Associated and choose the Key Value Map. |
4. On the Edit Key Value Map page, update the Value with the instance token value for the corresponding open connector instance. |
5. Provide the Org ID and User Secret from your corresponding Open Connector subscription.
**Actions Required on Configurations**

Depending on the configurations you have on your source system, you must configure the following in your target system:

- Custom IDP Setup (if any)
- Default role assignment to users
- Custom Role creation and its assignments
- Cloud Connector setup
- Principal Propagation setup at the subaccount level
- Changes to Principal Propagation policy for on-premise connectivity
- Migration of route service bindings. For more information, see Migrating Route Service Binding [page 646]
- Any integrations with other systems (like SAP Web IDE)
- Any other configurations that you created for API Management at the subaccount level of your source system

To know more about the entities that are cloned and the entities that aren’t cloned, see Cloned and Uncloned Entities [page 642].

**Note**

For the on-premise APIs, the URL of the target.basepath changes while migrating from Neo to Cloud Foundry. If you’ve customized any of the policies, where the target.basepath is being used, then make sure that you update the content of the policy accordingly in the target Cloud Foundry system. For example, after migration the target.basepath URL in Cloud Foundry might have an additional segment. You need to verify if this additional segment adversely affects the policy execution in target Cloud Foundry system.

**Migrating Route Service Binding**

If you’ve used the Route Service plan [page 51] to manage your Cloud Foundry applications, you can now migrate the existing route service binding, from the API Management instance on Neo to the new API Management instance on Cloud Foundry.

**Prerequisites**

- A route service binding exists between your application on Cloud Foundry and the API Management service instance in the Neo environment.
- You have enabled API Management on your Cloud Foundry sub account
- You have the space developer role assigned to you.

Depending upon the location of your application, and your API Management service instance, the steps to migrate the route service binding vary.

### Cloud Foundry Application and API Management subscription on the same subaccount

If your cloud foundry application and the API Management subscription are on the same sub account, then use the following steps to migrate the route service binding:

1. Create an API Management, API portal service instance using the service plan, apim-as-route-service. For more information, see Creating an API Management, API portal Service Instance [page 52]
2. Unbind your application from the API Management service instance on Neo. For more information, see Unbinding a Cloud Foundry Application from an API Management, API portal Service Instance [page 53]

3. Bind your application to the API Management service instance on Cloud Foundry. For more information, see Binding a Cloud Foundry Application to an API Management, API portal Service Instance [page 53]

**Cloud Foundry Application and API Management subscription on different sub accounts**

If your Cloud Foundry application and the API Management subscription are on different sub accounts, then use the following steps to migrate the route service binding:

1. Create a User Provided Service in the sub account where your Cloud Foundry application is present, using the proxy URL from the sub account in which your API Management instance is present. In order to create this User Provided Service, open the command prompt and use the following command

   ```
   cf create-user-provided-service apim-route-service -r https://apiproxy.url.from.source.
   OK
   ```

   For more information, see User Provided Service

2. Unbind your application from the API Management service instance on Neo. For more information, see Unbinding a Cloud Foundry Application from an API Management, API portal Service Instance [page 53]

3. Bind the User Provided Service created in the first step to the Cloud Foundry Application. For this binding, use the following command:

   ```
   cf bind-route-service cfapps.eu10.hana.ondemand.com --hostname <your-app-host>  apim-route-service
   ```

**Validate Your Target API Management System**

Validate that all your API Management artifacts have been cloned to the target system and that all your artifacts and route bindings are in working condition.

**Switch Over from Source to Target System**

You can choose to switch over completely from your source to target system after you’ve successfully cloned all the entities, performed the post-cloning tasks, and validated that your target system is working correctly.
This section explains the various scenarios for a switch-over:

### Switching Over Runtime Proxy URLs

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Actions Required for Switchover</th>
</tr>
</thead>
</table>
| If you want to retain the same proxy URL as that of your source system  | If the proxy URL of your source system is on a domain managed by SAP  
  There’s no option to retain the old proxy URL.  
  You must adopt the new proxy URL that is generated for your target system.  
  If the proxy URL of your source system is on a custom domain  
  1. Update the virtual host of the target system to that of the source system.  
  2. Perform a DNS change from the old cluster to a new cluster. |
| If you have multiple virtual hosts configured on your source system       | 1. Create multiple virtual hosts on your target system.  
  2. Bind each API proxy to the desired virtual host on your target system. |

### Switching Over Design time URLs of API portal and Developer portals

- Domains managed by SAP can’t be switched over.
- To switch over a custom domain, create an incident on the component OPU-API-OD-OPS through the SAP Support Portal.

### 2.4 Recommendations

This topic lists the recommendations that you must consider for migration.

- After cloning your API Management entities from the source to the target system for the first time, you must maintain both the systems, until you switch over from the source system to your target system completely.
- It is recommended that you always add or modify your entities in the source system, and clone it to the target system by rerunning the tool as and when required, instead of adding them directly to the target system.
2.5 Security

Security features of the Tenant Cloning tool.

Auditing and Logging

- The Tenant Cloning tool calls APIs provided by API portal and API business hub enterprise. Hence, there are no security-related events available in the tool.
- All application logs generated from the cloning tool are stored in “APIM-Tenant-Cloning-Tool.log”, an autogenerated log file.

Data Protection and Privacy

- The tool doesn’t persist any data on its own, nor is there a persistence layer.
- The tool logs the cloning status in the log file and in the output excel file named “apim-tct-output.xlsx”.
  - The log and output excel file contain e-mail IDs of application developers, needed for troubleshooting and migration reporting, which are being cloned from source to target system.
  - The tool doesn’t store any personal data (except e-mail IDs of application developers) in the log and output excel file.
  - We recommend storing the log file and output excel file securely, if further processing is needed; else these files must be deleted.
- The tool doesn’t read any sensitive personal data.
- The tool doesn’t change any personal data.

Identity and Access Management

- No specific identity and access management configuration is needed to run the tool.
- Application developer’s details are copied from source to target system as is. If some of the developer information isn’t valid in the IDP configured in the target tenant, it must be corrected.

Network and Communication Security

The tool uses standard HTTPS communication to make API calls, as provided by the API portal and API business hub enterprise.
3 Migrating API Management Subscription Created Using the Starter Plan Service Instance

You can choose to migrate the design-time components that you have in the Neo environment, which was previously set up using Starter Plan instance, to the Cloud Foundry environment, keeping the runtime components as is.

**Context**

You can also enable the new API Management design time subscription on the same Cloud Foundry subaccount, where you have created the starter plan service instance.

**Note**

You must subscribe to the API portal and the Developer Portal in the same Cloud Foundry subaccount where the starter plan instance is created.

Tenant type (for example, production and test) of the newly onboarded API Management on the Cloud Foundry environment must be same as that of the source API Management on the Neo environment.

**Caution**

The migration of the Starter Plan Service Instance might involve downtime of the API runtime calls.

**Procedure**

1. Raise a ticket through the SAP Support Portal. For more information, see Product Support.

   Use the following component for your incident:

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPU-API-OD-OPS</td>
<td>SAP API Management Operations - On Demand</td>
</tr>
</tbody>
</table>

   When submitting the incident, include the following information:
   - Incident title: Starter Plan Migration
   - Description: State that you want to migrate API Management subscription created using the Starter Plan.
   - Provide the Neo account details where API Management is enabled.
   - Provide the Cloud Foundry account details where starter plan service instance is created.
2. Prepare the target system by enabling the API Management subscription on the Cloud Foundry subaccount where your starter plan instance was created.

To complete the checks, before you start migrating your API Management artifacts nondisruptively from your source system to a target system, see Prerequisites [page 624].

3. Run the Tenant Cloning Tool in the DEFAULT stage. See Clone API Management Artifacts [page 626] for more information.

For the list of cloned and uncloned entities, see Cloned and Uncloned Entities [page 642]. For understanding the behavior of the Tenant Cloning Tool with respect to cloning some of the entities from your source system, see Tenant Cloning Tool Behavior [page 643].

4. After completing the cloning process, you must perform some actions, checks, and validations. For the task details, see Post Cloning Tasks [page 644].

For recommendations for migration, refer the following topic: Recommendations [page 648]

To know more about the security features of the tenant cloning tool, see Security [page 649].

5. Run the Tenant Cloning Tool in the SWITCHOVER stage. For more information, see Clone API Management Artifacts [page 626].

6. Inform SAP that migration is complete by updating the same ticket.

Results

Migration of API Management subscription created using the Starter Plan service instance is complete. There can be downtime for certain API proxies (having policies that are specific to Neo/Cloud Foundry environment) created out of on-premise providers.
Migrating API Management Subscription Created Using the Starter Plan Service Instance to Different Subaccounts

Migrate the design-time components from the Neo environment, which was previously set up using Starter Plan instance, to the Cloud Foundry environment, keeping the runtime components as is.

Context

With the Integration Suite premium edition license available in a different subaccount, you can migrate the API Management design time subscription to this subaccount as well.

i Note

Make a note of the following:

- Analytics data can’t be retained, as Advanced Analytics gets newly configured in the different subaccount. However, if you come across any analytics data from the previous subaccount, you must ignore the data and consider the analytics data after the migration task is completed.
- Subscribe to the API portal and the API business hub enterprise in the other Cloud Foundry subaccount. This is the subaccount with the Integration Suite premium edition license.
- Tenant type (for example, production and test) of the newly onboarded API Management on the Cloud Foundry environment must be same as that of the source API Management on the Neo environment.
- Ensure that both the source and the target subaccounts are in the same data center for migrating the API Management subscription to a different subaccount.
- To migrate to a different subaccount, you must provide the input `cfSubaccountTenantID` in the `apim-tct-input.json` file. For more information, see Clone API Management Artifacts [page 626].

⚠️ Caution

The migration of the Starter Plan Service Instance may involve downtime of the API runtime calls.

Procedure

1. Raise a ticket through the SAP Support Portal. For more information, see Product Support.

Use the following component for your incident:

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPU-API-OD-OPS</td>
<td>SAP API Management Operations - On Demand</td>
</tr>
</tbody>
</table>
When submitting the incident, include the following information:

- Incident title: Starter Plan Migration to Different Subaccounts
- Description: State that you want to migrate API Management subscription created using the Starter Plan to different subaccounts.
- Provide the Neo account details where API Management is enabled.
- Provide the Cloud Foundry account details where starter plan service instance is created.
- Provide the details of the target Integration Suite subscription account for migrating the starter plan subscription to a different subaccount. Since you already have the Integration Suite premium license available, you can create the API Management subscription before creating the ticket.

**Note**
Wait for SAP’s confirmation on the ticket before continuing with the migration.

2. To avoid any disruption, complete the checks before you start migrating your API Management artifacts from your source system to your target system. For details, see the Prerequisites [page 624] section.

3. Run the Tenant Cloning Tool in the DEFAULT stage. See Clone API Management Artifacts [page 626] for more information.

   For the list of cloned and uncloned entities, see Cloned and Uncloned Entities [page 642]. For understanding the behavior of the Tenant Cloning Tool with respect to cloning some of the entities from your source system, see Tenant Cloning Tool Behavior [page 643].

   **Note**
   Since this is starter plan migration scenario, only the API portal artifacts at this stage get cloned.

4. After completing the cloning process, you must perform some actions, checks, and validations. For the task details, see Post Cloning Tasks [page 644].

   For recommendations for migration, refer the following topic: Recommendations [page 648]

   To know more about the security features of the tenant cloning tool, see Security [page 649].

5. Run the Tenant Cloning Tool in the SWITCHOVER stage. For more information, see Clone API Management Artifacts [page 626].

   **Note**
   If applicable, API business hub enterprise entities are cloned in this step.

6. Inform SAP that migration is complete by updating the same ticket.

**Results**

Migration of API Management subscription (created using the Starter Plan service instance) to a different subaccount is complete. There can be downtime for certain API proxies (having policies that are specific to Neo/ Cloud Foundry environment) created out of on-premise providers.
# Glossary

## Terms related to API Management

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Management</td>
<td>• Creates simple digital experiences for your consumers, partners, and employees.&lt;br&gt;• Uses a technology that helps you to share digital assets and enable developer communities to consume these assets in new channels, devices, and user interfaces.&lt;br&gt;Available in the cloud, the technology helps promote co-innovation among employees, partners, and the developer community.&lt;br&gt;• Reduces complexity by leveraging a single provisioning platform (API Platform) to provide unified access and governance of APIs across a heterogeneous landscape.&lt;br&gt;• Provides one experience for managing and monitoring all APIs across various data platforms and is enriched with real-time analytics and enables consumers to access relevant data directly in a secure manner. Selective data can be exposed while reducing the risk of security breaches.</td>
</tr>
<tr>
<td>SAP API Management</td>
<td>It lets you publish, promote, and oversee APIs in a secure and scalable environment</td>
</tr>
<tr>
<td>Neo environment</td>
<td>SAP BTP. Neo environment contains SAP propriety runtime. Neo is a feature-rich and easy-to-use development environment and allows you to develop Java, SAP HANA XS, and HTML5 applications.&lt;br&gt;The Neo environment uses virtual machines, allowing you to install and maintain your own applications in scenarios that aren’t covered by the platform.</td>
</tr>
<tr>
<td>Cloud Foundry environment</td>
<td>SAP BTP. Cloud Foundry environment contains the Cloud Foundry Application Runtime, which is based on the open-source application platform managed by the Cloud Foundry Foundation&lt;br&gt;Application developers can use the Cloud Foundry environment to enhance SAP products and to integrate business applications, as well as to develop entirely new enterprise applications based on business APIs that are hosted on SAP BTP.</td>
</tr>
<tr>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>The Cloud Foundry environment</td>
<td>The Cloud Foundry environment allows you to use multiple programming languages such as Java, Node.js, and community/bring-your-own language options.</td>
</tr>
<tr>
<td>API Platform</td>
<td>Provides tools to manage APIs and it facilitates the inclusion of new APIs, configuration of existing APIs, and helps you manage developers and apps. It also helps you create and consume APIs, whether you want to build API proxies as a service provider or use APIs, SDKs, and other convenient services as an app developer.</td>
</tr>
<tr>
<td>API Analytics</td>
<td>Provides powerful analytical tools to track your API usage. Use the API analytics to collect information on the IP, URL, user ID for API call information, latency data, and so on.</td>
</tr>
<tr>
<td>Developer Services</td>
<td>Provides tools to manage app developers. Provides the ability to onboard developers and creates a developer portal for publicly available products.</td>
</tr>
<tr>
<td>API Management Account</td>
<td>An API Management account is the highest level of data hierarchy. An account is a representation of all components including APIs, products, applications, systems, users, and developers.</td>
</tr>
<tr>
<td>System</td>
<td>In API Management System refers to the API provider systems where the actual backend services reside. The system could either be an ABAP system, SAP Gateway system, Enterprise Services Repository, or systems that host generic REST services or third-party provider systems. API Management allows you to add and manage an API provider system. After you have added a system, you can browse for the APIs in that system.</td>
</tr>
<tr>
<td>User</td>
<td>API Management can have multiple users. Different users have different roles and privileges assigned. For example, people who create APIs and products or analyze the metrics or the application consumer who can access the APIs provisioned by API Management.</td>
</tr>
<tr>
<td>API</td>
<td>APIs are Application Programming Interfaces. They comprise a set of routines, protocols, and tools for building software applications. APIs define sets of requirements that govern how applications communicate with one another. They facilitate interaction by selectively exposing certain functionalities, allowing different applications, websites, or devices to communicate effectively with each other.</td>
</tr>
<tr>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Product</strong></td>
<td>A product is a bundle of APIs. It contains metadata specific to your business for monitoring or analytics. For example, all APIs related to CRM can be bundled as one CRM product. API Management collects data for analyzing the products.</td>
</tr>
<tr>
<td><strong>Developer</strong></td>
<td>One or more developers can create applications in the API Management account. A developer can consume APIs but cannot create APIs. To create an application, the developer must have registered the account. After having created an application, the developer uses the app(application) key to consuming the APIs.</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Applications include the Web or mobile applications that consume the exposed APIs. When you create an application, you select the product to include in this application. For each application that you create, API Management generates an app key and secret. Use this key to gain access to multiple products. Developers create one or more applications using the APIs you expose.</td>
</tr>
<tr>
<td><strong>App Key</strong></td>
<td>Based on the authorization mechanism you define for your APIs; the application passes an app (application) key together with every request to your APIs. If that key is valid, the request is permitted. API Management supports different types of authentication, such as a simple API key, OAuth, and so on.</td>
</tr>
<tr>
<td><strong>API Portal</strong></td>
<td>You can browse through this API package for API Admin services with the required resources.</td>
</tr>
<tr>
<td><strong>Developer Portal</strong></td>
<td>You can browse through this API package for application development services that are offered.</td>
</tr>
<tr>
<td><strong>Metering</strong></td>
<td>You can now browse through this API package to view metering data for APIs, API Products, and applications in API Portal.</td>
</tr>
<tr>
<td><strong>API Analytics</strong></td>
<td></td>
</tr>
<tr>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Client SDK</td>
<td>A client software development kit (SDK) is available for developers through a non-commercial license on open source sites.</td>
</tr>
<tr>
<td></td>
<td>On the API Portal home page, choose the Client SDK. On selecting the client SDK, you are navigated to the maven repository, where you can download this package.</td>
</tr>
</tbody>
</table>
# Glossary-2

Terms related to API Management

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze APIs</td>
<td>Analyzing APIs helps you know more about API traffic data, and you can trace the API calls with real-time insights from your data.</td>
</tr>
<tr>
<td>API</td>
<td>API stands for Application Programming Interface. An API connects different software components by facilitating interaction with applications, websites, or devices to communicate effectively with each other by allowing data sharing.</td>
</tr>
<tr>
<td>API Analytics</td>
<td>API Analytics provides powerful analytical tools to track your API’s health and usage. Using API Analytics, you get to utilize the sample analytical charts and key performance indicators (KPIs). These charts and KPIs are preconfigured in the dashboard.</td>
</tr>
<tr>
<td>API Call</td>
<td>An API call is a request made to the server using APIs.</td>
</tr>
<tr>
<td>API Gateway</td>
<td>An API Gateway is an execution engine that intercepts the API calls and executes the policies. The API Gateway adds on to the capabilities of API Management in security handling, traffic management, and governance.</td>
</tr>
<tr>
<td>API Design</td>
<td>An API design is used to provide an efficient interface and helps customers in a better understanding of the use-case of a product.</td>
</tr>
<tr>
<td>API Designer</td>
<td>An API Designer is a person who designs APIs. In the design phase, an API designer can define the requirements for APIs and plan the services (services like ODATA or REST) that can be used to expose to customers.</td>
</tr>
<tr>
<td>API Management Account</td>
<td>An API Management account is the highest level of data hierarchy. This account hosts the API platform, which is a combination of API and API business hub enterprise.</td>
</tr>
<tr>
<td>API Platform</td>
<td>API Platform enables enterprises to stimulate innovation, implement distributed services and data, adapts to market and customer needs. Hence, APIs have become the foundation of the fast-moving digital economy.</td>
</tr>
<tr>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>API Portal</td>
<td>An API Portal provides tools to manage APIs, facilitates the inclusion of new APIs, and helps you manage them. It also helps you create APIs, whether you want to build API proxies as a service provider or use APIs, SDKs, and other convenient services as an app developer.</td>
</tr>
<tr>
<td>API Policy</td>
<td>API Management provides capabilities to define the behavior of an API by using 'policies.' A policy is a program that executes a specific function at runtime. They provide the flexibility to add common functionalities on an API without having to code them individually each time. Policies provide features to secure APIs, control the API traffic, and transform message formats. You can also customize the behavior of an API by adding scripts and attaching them to policies.</td>
</tr>
<tr>
<td>API Provider</td>
<td>An API provider is an abstraction for a system that defines the connection details for services running on specific hosts whose details you want to access. It also displays data through a programmatically consumable service or an API.</td>
</tr>
<tr>
<td>API Proxy</td>
<td>An API proxy is a masked URL that disconnects the app-surfacing API from your backend services, protecting those apps from backend code changes.</td>
</tr>
<tr>
<td>Application</td>
<td>Applications include the Web or mobile applications that consume the exposed APIs. When you create an application, you select the product to include in this application. For each application that you create, API Management generates an app key and secret. Use this key to gain access to multiple products.</td>
</tr>
<tr>
<td>Application Key</td>
<td>An application key is an encryption code assigned to a specific application. Different applications have different application keys. This key validates if the API requested was associated to a particular application. After having created an application, the developer uses the application key to consume the APIs.</td>
</tr>
<tr>
<td>Build APIs</td>
<td>Build API proxies prominently and configure API policies as steps in the API flow. Customize API behavior using code. Plus, modify from or to any protocol.</td>
</tr>
<tr>
<td>Client SDK</td>
<td>A client software development kit (SDK) is available for developers through a noncommercial license on open-source sites.</td>
</tr>
<tr>
<td></td>
<td>On the API Portal home page, choose the Client SDK. On selecting the client SDK, you are navigated to the maven repository, where you can download this package.</td>
</tr>
<tr>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Consume API</td>
<td>Consume or use APIs via the API business hub enterprise. In the API business hub enterprise, an application developer registers, explores the API exposed by customers, creates applications, and tests APIs.</td>
</tr>
<tr>
<td>Cloud Foundry Environment</td>
<td>SAP BTP Cloud Foundry environment contains the Cloud Foundry Application Runtime, which is based on the open-source application platform managed by the Cloud Foundry Foundation.</td>
</tr>
<tr>
<td>API business hub enterprise</td>
<td>A personalized portal that lets you to instantly explore, test, get API keys, and innovate quick. You can also accelerate API adoption with offerings, rate limits, and pricing.</td>
</tr>
<tr>
<td>Developer Services</td>
<td>Services that provide tools to manage app developers. Also, provide the ability to onboard developers and creates a API business hub enterprise for publicly available products.</td>
</tr>
<tr>
<td>Keystore</td>
<td>A keystore contains the SSL Certificate and private key that is used to identify the entity during SSL Handshake.</td>
</tr>
<tr>
<td>Monetize APIs</td>
<td>Allows the API Providers to generate revenue via a rate plan service or billing service for the API usage.</td>
</tr>
</tbody>
</table>

**N-Z**

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEO Environment</td>
<td>SAP BTP NEO environment contains SAP propriety runtime. NEO is a feature-rich and easy-to-use development environment, allowing you to develop Java, SAP HANA XS, and HTML5 applications.</td>
</tr>
<tr>
<td>Policy Template</td>
<td>A policy template is a policy structure designed to define the behavior of an API policy. You can create, apply, update, import, export, and delete a policy template using the API Portal.</td>
</tr>
<tr>
<td>Publish API</td>
<td>Publish the API product on the API portal. Make it available for consumption.</td>
</tr>
<tr>
<td>Product</td>
<td>A product is a package that contains APIs and metadata specific to your business for monitoring or analytics. For example, all APIs related to CRM can be bundled as one CRM product.</td>
</tr>
<tr>
<td>Rate Plan</td>
<td>Rate Plan is the price per API call made to any external app service. You can decide the basic charge, mode of currency, and rate plan type while creating a rate plan for all the API calls made.</td>
</tr>
<tr>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SAP API Management</td>
<td>A service that offers you a harmonized experience for creating, maintaining, governing, and monitoring all your APIs across data-platforms. You can also leverage real-time analytics to make quick business decisions that are critical in today’s API-first strategy.</td>
</tr>
<tr>
<td>System</td>
<td>In API Management, a system refers to the API provider system where the actual backend services reside. The system could either be an ABAP system, SAP Gateway system, Enterprise Services Repository, or systems that host generic REST services or third-party provider systems. For example, API Management allows you to add and manage an API provider system. After you have added a system, you can browse for the APIs in that system.</td>
</tr>
<tr>
<td>Test Console</td>
<td>API Management provides an API Test Console, which enables you to test your APIs. Testing an API is essential to understand the runtime behavior of the APIs. The test console allows you to explore the resources associated with an API and execute the operations.</td>
</tr>
<tr>
<td>Trust Store</td>
<td>A truststore contains certificates used to validate certificates obtained as part of SSL handshaking.</td>
</tr>
<tr>
<td>User</td>
<td>API Management can have multiple users. Different users have different roles and privileges assigned to them. For example, people who create APIs and products or analyze the metrics.</td>
</tr>
</tbody>
</table>
Reuse Content for CF plans

Deleting an API Management Service Instance

Delete an API Management service instance.

Prerequisites

- You have the space developer role assigned to you.
- You have created an API Management service instance.

Use the following procedure to delete an API Management service instance on Cloud Foundry.

Procedure

1. In your Web browser, open the SAP Cloud BTP Cockpit.
2. In the provider account, choose Services > Service Marketplace > Instances.
3. Select the API Management service tile.
4. From the list of instances visible, select the instance that you want to delete and choose 🔴.
5. Choose OK.

Updating an API Management Service Instance

Update user credentials for an API Management service instance.

Prerequisites

- You have created an API Management service instance.
- You have logged on as a space developer.

Context

Perform the following steps to update user credentials for an API Management service instance.

Procedure

Open the command-line interface for Cloud Foundry and enter the following command:

```bash
Note
You can update a service only from the command-line interface and not from SAP Cloud BTP cockpit.

Sample Code

cf update-service apim-service-instance-name -c '{"apiportal_admin": "<api portal admin id>", "apiportal_password": "<api portal password>", "consent": <value should be true>}'
```
For more information on Updating a service, see Update Service.

### Binding a Cloud Foundry Application to an API Management Service Instance

Create a service instance and bind the CF application to API management. When you bind an application, an API proxy is created and a new route is added to the application. The route initially redirects all calls to the proxy URL and then to the application.

**Prerequisites**

- You have created an API Management service instance.
- You have logged on as a space developer.

Open the command-line interface for Cloud Foundry and enter the following command:

```bash
cf bind-route-service sap-cf-domain.com apim-service-instance-name --hostname my-app -c '{"api_name": "custom_api_proxy_name"}'
```

**Example**

```bash
cf bind-route-service cfapps.sap.hana.ondemand.com apim-prod-instance --hostname taxapp
```

Cloud foundry URL for the above example is `https://taxapp.cfapps.sap.hana.ondemand.com`.

```bash
cf bind-route-service sap-cf-domain.com apim-service-instance-name --hostname my-app -c '{"api_name": "test_api"}'
```

With parameters for Windows system

```bash
cf bind-route-service sap-cf-domain.com apim-service-instance-name --hostname my-app -c '{"api_name": "test_api"}'
```

**Note**

API Management supports only English alpha numeric, hyphens (-) and underscores (_) characters for "api_name".

You can bind an application to a service only from the command-line interface and not from SAP BTP Cockpit.

Providing a value for the parameter during binding is optional. If you provide a value for api_name, then the API proxy created in API portal for current binding gets the given name. Also, if an API with the same name
exist in the API portal, then the same API proxy is used for the binding. That is, the API proxy end point is registered as the route service URL for the current binding.

For more information on binding an application, see bind route service.

Unbinding a Cloud Foundry Application from an API Management Service Instance

Unbind an application to an API Management by deleting the service instance. When you unbind an application, API proxy is eliminated from the application.

Prerequisites
- You have logged on as a space developer
- You have bound an application to an API Management service instance.

Open the command prompt and enter the following command:

```
# Sample Code

cf unbind-route-service sap-cf-domain.com apim-service-instance-name --hostname my-app

<-- Example

cf unbind-route-service cfapps.sap.hana.ondemand.com apim-prod-instance --hostname taxapp

-->
```

i Note

You can unbind an application from a service only from the command-line interface and not from SAP BTP Cockpit.

For more information on unbinding an application, see Unbind route service.
Important Disclaimers and Legal Information

Hyperlinks

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

About the icons:

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

- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

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

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

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

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.