Oracle® Cloud

Using the REST Adapter with Oracle Integration 3

F45599-40
June 2025

Oracle Cloud Using the REST Adapter with Oracle Integration 3,

F45599-40

Primary Author: Oracle Corporation

Contents

Preface

1 Understand the REST Adapter

2 REST Adapter Concepts

3 Create a REST Adapter Connection

4 Add the REST Adapter Connection to an Integration

5 Implement Common Patterns Using the REST Adapter

Preface

This guide describes how to configure this adapter as a connection in an integration in Oracle Integration.

> **Note:**
>
> The use of this adapter may differ depending on the features you have, or whether your instance was provisioned using Standard or Enterprise edition. These differences are noted throughout this guide.

Topics:

- Audience
- Documentation Accessibility
- Diversity and Inclusion
- Related Resources
- Conventions

Audience

This guide is intended for developers who want to use this adapter in integrations in Oracle Integration.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `https://www.oracle.com/corporate/accessibility/`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `https://support.oracle.com/portal/` or visit `Oracle Accessibility Learning and Support` if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Resources

See these Oracle resources:

- Oracle Cloud at `http://cloud.oracle.com`
- *Using Integrations in Oracle Integration 3*
- *Using the Oracle Mapper with Oracle Integration 3*
- Oracle Integration documentation on the Oracle Help Center.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Understand the REST Adapter

Review the following conceptual topics to learn about the REST Adapter and how to use it as a connection in integrations in Oracle Integration. A typical workflow of adapter and integration tasks is also provided.

Topics:

- REST Adapter Capabilities
- REST Adapter Restrictions
- REST Adapter Use Cases
- Workflow to Create and Add a REST Adapter Connection to an Integration

REST Adapter Capabilities

The REST Adapter can expose integrations as REST APIs by configuring a REST Adapter connection as a trigger. The REST Adapter can also consume any external REST API by configuring a REST Adapter connection as an invoke. This section identifies the capabilities of the REST Adapter when used as a trigger or invoke connection.

> **Note:**
>
> The REST Adapter treats all endpoints as they are exposed. The REST Adapter does not filter or change any of the APIs exposed by the application to which you are connecting. If there is a native adapter for the application to which you are connecting, use that adapter instead. If you choose to use the REST Adapter instead of the native adapter, the API restrictions and deprecation policies apply as specified in the respective application’s documentation.
> To connect to the Oracle HCM Cloud SOAP APIs, see Oracle HCM Cloud Adapter Capabilities.

- Capabilities When Configuring a Trigger Connection to Expose an Integration as a REST API
- Capabilities When Configuring an Invoke Connection to Consume External REST APIs

Capabilities When Configuring a Trigger Connection to Expose an Integration as a REST API

The REST Adapter provides the following capabilities when configured as a trigger connection to expose an integration as a REST API. When configured as a trigger connection, the REST Adapter lets your integration receive inbound messages from an application.

- Expose REST Endpoints
- Upload Payload Formats

- Configure Endpoint Properties
- Enforce Incoming Message and Attachment Size Limitations
- Support Multipart/Mixed and Multipart/Form-Data Types
- Support Swagger
- Expose Standard and Custom HTTP Headers
- Propagate User Identity to Oracle NetSuite

Expose REST Endpoints

You can expose a REST endpoint that can accept the request and process it asynchronously.

Upload Payload Formats

You can upload the following:

- Complex XML schema definitions as a zipped archive to define data definitions for XML content during REST Adapter configuration. See Complex Schema Support.
- Sample XML documents to define data definitions for XML content during REST Adapter configuration. The following XML documents are supported for schema generation:
  - XML with no namespace.
  - XML with a homogenous namespace.
  - XML files up to 3 MB in size.
- Schemas with simpleTypes having the restriction facet enabled. Both named and anonymous types are supported.

Configure Endpoint Properties

You can configure the following:

- Relative resource URI.
- HTTP methods GET, PUT, POST, DELETE, and PATCH.
- Template and query parameters.
- Request/response payload.
  - Supports JSON, XML, binary (inline and unstructured), and URL-form-encoded payloads.
  - Supports homogenous JSON arrays including top-level arrays.
  - Supports multidimensional JSON arrays (see Homogenous Multidimensional Array Support in JSON Documents).
- REST APIs exposed using the REST Adapter are secured using Basic Authentication, OAuth token-based authentication, and JWT-based authentication.
- REST APIs implement the HTTPS protocol, thereby enforcing all incoming requests to have transport level security.
- Multiple operation entry points with different resource URIs and HTTP actions/verbs. Each operation represents a different pick action branch in a single integration. This feature eliminates the need to create multiple integrations (each with a separate resource URI and verb) to perform different operations. See Receive Requests for Multiple Resources in a Single REST Adapter Trigger Connection of *Using Integrations in Oracle Integration 3*.

- REST APIs exposed using the REST Adapter to be CORS-compliant (see Cross-Origin Resource Sharing (CORS)).
- Sample cURL syntax generation on the Summary page of the Adapter Endpoint Configuration Wizard for the configuration options that you have selected during REST Adapter connection configuration, such as security policy, headers, parameters, and so on. See Summary Page.

Enforce Incoming Message and Attachment Size Limitations

Support is provided for the following:

- Ensures that incoming (trigger) message requests without attachments do not exceed the size limit. See Service Limits in *Provisioning and Administering Oracle Integration 3*.

  Messages with attachments (for example, multipart/mixed and multipart/form-data) are not subject to this constraint. If the size of the structured message (for example, XML/JSON) exceeds the size limit, an HTTP error code message is returned to the client: `413 Request entity too large`.
- Ensures that incoming (trigger) JSON attachments do not exceed the size limit. If the size of the JSON attachment exceeds the limit, an HTTP error code message is returned to the client: `413 Request entity too large`. See Service Limits in *Provisioning and Administering Oracle Integration 3*.
- Ensures that incoming (trigger) structured message payload requests (any content-type header containing JSON, XML, HTML, YAML, or YML) from a client do not exceed the size limit. If the size of the structured message exceeds these values, an HTTP error code message is returned to the client: `413 Request entity too large`.

  See Service Limits in *Provisioning and Administering Oracle Integration 3*.

Support Multipart/Mixed and Multipart/Form-Data Types

Support is provided for the following:

- Consumption and publication of OpenAPI with multipart/mixed and multipart/form-data in REST Adapter trigger connections. See Consume and Publish OpenAPI Documents with Multipart/Mixed and Multipart/Form-Data.
- Multipart attachments (content-types: multipart/mixed, and multipart/form-data) in request/response messages while creating an integration to expose a REST endpoint that accepts incoming request messages with multipart attachments and/or sends responses with multipart attachments (see Multipart Attachment Support for Trigger and Invoke Connections).

Support Swagger

Automatic production of a Swagger 2.0–compliant document for REST APIs exposed using the REST Adapter is supported. This document describes the metadata for the generated REST APIs.

Expose Standard and Custom HTTP Headers

Standard and custom HTTP headers to model an integration are supported to expose standard and custom HTTP header properties to Oracle Integration for downstream processing (see Standard and Custom Header Support).

Propagate User Identity to Oracle NetSuite

Support is provided for creating custom JWT claims to propagate the user identity from the REST Adapter to Oracle NetSuite. Identity propagation enables you to securely transfer the same token ID and token secret for the user to Oracle NetSuite. See Propagate User Identity from the REST Adapter to Oracle NetSuite with Custom JWT Claims.

Capabilities When Configuring an Invoke Connection to Consume External REST APIs

The REST Adapter provides the following capabilities when configured as an invoke connection to consume external REST APIs. When configured as an invoke connection, the REST Adapter sends messages to a target application endpoint.

- Enforce Outgoing Message and Attachment Size Limitations
- Publish OpenAPI Documents
- Invoke Oracle Cloud Infrastructure REST APIs
- Connect to Private Resources
- Consume REST APIs
- Invoke Amazon Web Services (AWS)
- Configure Endpoint Properties
- Upload Payload Formats
- Support Multipart/Mixed and Multipart/Form-Data Types
- Support Resource Principal Session Tokens
- Support JWT Client and User Assertions
- Support Obfuscation of Sensitive Data
- Expose Standard and Custom HTTP Headers
- Invoke Co-located REST APIs
- Dynamically Change Endpoints

Enforce Outgoing Message and Attachment Size Limitations

Support is provided for the following:

- Ensures that responses containing attachments for outbound REST requests do not exceed 1 GB. These attachments can be multipart/mixed, multipart/form-data, or application/octet-stream. If the attachment exceeds 1 GB, an HTTP error code message is returned: `413 Request entity too large`
- Ensures that outgoing (invoke) messages returning an unstructured payload (multipart-formdata and binary/octed-stream) from a client do not exceed the size limit.
- Ensures that outgoing (invoke) messages returning structured message payloads (any content-type header containing JSON, XML, HTML, YAML, or YML) to a client do not exceed the size limit.

  See Service Limits in *Provisioning and Administering Oracle Integration 3*.

Publish OpenAPI Documents

You can publish REST-based integrations as OpenAPI documents. OpenAPI support is available when configuring the REST Adapter as an invoke connection. You provide a link to the OpenAPI document to publish or consume.

OpenAPI support enables you to perform the following tasks:

- - Publish an OpenAPI document describing an Oracle Integration REST endpoint. You can invoke the REST endpoint with the published document using a REST client such as postman.
    See Publish REST-Based Integrations as OpenAPI Documents.
  - Consume an OpenAPI document using the REST Adapter.

  OpenAPI-defined headers are automatically supported except for the standard headers that are currently disabled in the Adapter Endpoint Configuration Wizard.

Invoke Oracle Cloud Infrastructure REST APIs

You can invoke Oracle Cloud Infrastructure REST APIs such as Oracle functions, streaming, storage and so on as an integral part of Oracle Integration integration flows.

Connect to Private Resources

You can connect to private resources that are in your virtual cloud network (VCN) with a private endpoint. See Connect to Private Resources in *Provisioning and Administering Oracle Integration 3* and Configure the Endpoint Access Type. This type of connection does not use the connectivity agent. The REST Adapter does not support private endpoints with trigger connections. Only invoke connections are supported.

Consume REST APIs

Support is provided for the following:

- Consuming any REST API described using Swagger 2.0/RAML/OpenAPI documents and the Oracle Metadata Catalog. The REST Adapter can automatically discover and present the available resources and operations present in the documents for configurations. The metadata regarding operation-specific request and response messages available in the document is automatically made available for mapping and other activities.
- Consuming external REST APIs that are not described using Swagger/RAML/OpenAPI documents. You can declaratively specify the HTTP method and the sample JSON document/XML schema for describing the shape of the request and response messages.
- Consuming REST APIs protected using HTTP Basic Authentication, OAuth Client Credentials (two-legged flow), OAuth Resource Owner Password Credentials (two-legged flow), OAuth Authorization Code Credentials (three-legged flow), OAuth Custom Three Legged Flow, OAuth Custom Two Legged Flow, OAuth 1.0a One Legged Authentication, Amazon Web Services (AWS) Signature Version 4, and Oracle Cloud Infrastructure (OCI) Signature Version 1. There is also support for consuming APIs that are unprotected.
- Consuming external REST APIs that are protected using transport level security. The REST Adapter supports one-way SSL and two-way SSL. Oracle Integration supports a certificate management user interface to upload public certificates for external APIs that are protected either using lesser known certifying authorities (CA) or self-signed certificates.
- Consuming external REST APIs hosted on a two-way SSL server requiring client side (Oracle Integration) identity. Oracle Integration provides support for exchanging the client side identity with the server hosting the external API.

Invoke Amazon Web Services (AWS)

You can invoke external REST endpoints supporting the Amazon Web Services (AWS) Signature Version 4 authentication type. You can use the Amazon Web Services (AWS) Signature Version 4 security policy with the connectivity agent for scenarios in which you need to invoke AWS APIs hosted in an on-premises environment.

Configure Endpoint Properties

You can configure the following (see Configuration Parameters):

- Relative resource URI.
- HTTP methods GET, PUT, POST, DELETE, and PATCH.
- Template and query parameters.
- Request/response payload:
  - Supports JSON, XML, binary (inline and unstructured), and URL-form-encoded payloads.
  - Supports homogenous JSON arrays.
  - Supports multidimensional JSON arrays (see Homogenous Multidimensional Array Support in JSON Documents).
  - Supports delivery of form parameters as part of a request body.
- Sample cURL syntax generation on the Summary page of the Adapter Endpoint Configuration Wizard for the configuration options that you have selected during REST Adapter connection configuration, such as security policy, headers, parameters, and so on. See Summary Page.

Upload Payload Formats

Support is provided for uploading the following:

- Sample XML documents to define the data definition for XML content during REST Adapter configuration. The following XML documents are supported for generating the data definition:
  - XML with no namespace.
  - XML with a homogenous namespace.
  - XML files up to 3 MB in size.
- Sample JSON documents to define data definitions during REST Adapter configuration.
- Complex XML schema definitions as a zipped archive to define data definitions for XML content during REST Adapter configuration (see Complex Schema Support).
- Schemas with simpleTypes having the restriction facet enabled. Both named and anonymous types are supported.

Support Multipart/Mixed and Multipart/Form-Data Types

Support is provided for the following:

- Multipart attachments (content-type: multipart/mixed, and multipart/form-data) in request/ response messages in an integration while sending a request to an external REST endpoint that accepts incoming request messages with multipart attachments and/or sends responses with multipart attachments (see Multipart Attachment Support for Trigger and Invoke Connections).

Support Resource Principal Session Tokens

You can use the Resource Principal Session Token (RPST). RPST enables an Oracle Integration instance (the resource) to authenticate itself with and consume Oracle Cloud Infrastructure services, such as Oracle Cloud Infrastructure Functions, Oracle Cloud Infrastructure Object Storage, Oracle Cloud Infrastructure Vision, and more. See Resource Principal Session Token Support and RPST and OCI Service Invocation Security Policy Use.

Support JWT Client and User Assertions

JWT client and user assertions with the OAuth Client Credentials are supported using the JWT Client Assertion security policy and the JWT User Assertion security policy, respectively. JWT assertions enable you to invoke a service provider that does not regard an OAuth client secret as secure. Trust is established with a key pair exchange instead of a client secret. See JWT Assertion Support for Outbound Invocations and Invoke a Service Provider API with a JWT Assertion.

The OAuth using the JWT User Assertion security policy also provides support for OAuth identity propagation when invoking REST API operations. OAuth identity propagation enables you to securely transfer the same user identity and access credentials across services. See Propagate OAuth User Identity Between Services.

Support Obfuscation of Sensitive Data

The obfuscation of sensitive data such as client ID, client secret, scopes, API key, and others is supported when configuring fields for the OAuth Custom Three Legged Flow security policy and the OAuth Custom Two Legged Flow security policy. See Data Obfuscation Support.

Expose Standard and Custom HTTP Headers

Support is provided for the following:

- Accessing and setting standard and custom HTTP headers exposed by external REST APIs (see Standard and Custom Header Support).
- Using extensibility support to access plurality of OAuth 2 providers (see Standard and Custom Header Support).

Invoke Co-located REST APIs

You can invoke co-located REST APIs in an optimized manner.

The Oracle Integration runtime determines if the endpoint being invoked is co-located by checking if the endpoint URL has a load balancer address. If the endpoint URL has a load balancer address, the endpoint is considered co-located and the HTTP request is optimized by accessing the service locally using the non-SSL HTTP protocol.

Dynamically Change Endpoints

You can dynamically change the (invoke) outbound endpoint configuration (see Support for Dynamic REST Endpoints).

REST Adapter Restrictions

Note the following REST Adapter restrictions.

- The REST Adapter does not support private endpoints with trigger connections. Only invoke connections are supported. Attempting to use a trigger connection results in the following error:

  ```
  Connection with PrivateEndpoint access type is not supported for trigger.
  Please choose a valid
  access type for the trigger.
  ```

- The OCI Service Invocation security policy does *not* work in cloud tenancies that are not enabled for identity domains. If you don't know your cloud tenancy status, ask your administrator.
- Transport Layer Security (TLS) version 1.3 is not supported.
- REST endpoints can be protected using two-way SSL or mutual TLS authentication (mTLS). The REST Adapter supports accessing these endpoints. Authorization endpoints that procure and manage OAuth access tokens are also REST endpoints. However, these endpoints are not certified for use with the REST Adapter connection when protected using two-way SSL.
- Two-way SSL is not supported for calls to external services through the connectivity agent. Two-way SSL requires direct connectivity from Oracle Integration without the connectivity agent.
- The maximum permissible limit for JSON file samples is 100 KB.
- Plain/text content-type can be sent or received as unparsed content by the REST Adapter using the raw payload option.
- Consuming external REST APIs that are protected using NTLM or digest token-based authentication are not supported.
- When configuring the REST Adapter to work with the on-premises connectivity agent on the Connections page, only the **Basic Authentication**, **OAuth Client Credentials**, **OAuth Resource Owner Password Credentials**, **OAuth Custom Two Legged Flow**, and **No Security Policy** security policies in the **Security Policy** list are supported.
  The following security policies in the **Security Policy** list are *not* supported for use with the connectivity agent:
  - **AWS Signature Version 4**
  - **OAuth Authorization Code Credentials**
  - **OAuth Custom Three Legged Flow**
  - **API Key Based Authentication**
  - **OAuth 1.0 One Legged Authentication**
  - **OCI Signature Version 1**

  See On-Premises REST API Support with the Agent.
- The REST Adapter automatically encodes the value of query parameters before invoking a service. The REST Adapter has no way of knowing if you have already encoded a query parameter. Ensure that you assign *unencoded* values to query parameters. Assigning encoded values leads to double encoding.

  For example, assume query parameter `q` has the following value:

  ```
  q=a+b
  ```

  This may mean that the value of `q` was intended to be `a b`, but was encoded by the user.

The intention may also have been to send `a+b`, which must be URL-encoded as `a%2Bb` before sending.

- Polymorphic constructs are not supported when publishing OpenAPIs.
- Unicode characters in the range of `\u0000` to `\u001F` are control characters and are not allowed in JSON elements.
- You can customize the response status. However, this is not shown as part of the Swagger contract because runtime overrides are not known as part of the interface.
- HTTP response status cannot be customized under the following conditions:
  - If the request is asynchronous one way. This is because the response status is always `201`.
  - Errors that occur during trigger request/response handling are reported using a predefined error code.
  - Basic routing integrations (map my data integrations) don't allow fault handling and error responses in such scenarios.
  - If the HTTP response status is set to `204`, the response is sent back without any content.
- If you use the OAuth 1.0 One Legged Authentication security policy for integrations with Oracle NetSuite, ensure your REST Adapter connections use **HMAC-SHA256** or **HMAC-SHA1**. If you need to make an update, integration reactivation is not required. See Configure Connection Security.

> **Note:**
>
> There are overall service limits for Oracle Integration. A service limit is the quota or allowance set on a resource. See Service Limits.

Swagger/OpenAPI Restrictions

Not all Swagger constructs are understood by the REST Adapter. Note the following limitations when publishing and consuming Swagger/OpenAPI.

Publish Restrictions

Note the following publish restrictions for the REST Adapter.

- Polymorphic constructs are not supported when publishing OpenAPIs.
- Endpoints created using REST Adapter trigger connections are described using a Swagger and openAPI definition. REST endpoints having an XML request/response are not correctly described. Clients must not rely on the Swagger/openAPI definition.
- Each REST Adapter trigger connection endpoint is published as a Swagger document. The Swagger document usually has a single resource and verb.
  However, in the case of multiple verbs and resources, the Swagger document has multiple resources or paths in Swagger and multiple verbs for each resource.
- While OpenAPI lets you define dictionaries types where the keys are strings, dictionary types are *not* supported in Oracle Integration.
- Only JSON payload is supported for publishing. XML payload is not supported.

Consume Restrictions

The REST Adapter as a client can be configured with a Swagger definition based on which it can discover and list the existing resource. However, some Swagger operations cannot currently be correctly consumed by the REST Adapter:

- Only three multipart combinations are supported for consumption:
  - One or many file upload body parts and any number of plain text fields using multipart/form-data
  - One or many file upload body parts and one JSON body part using multipart/form-data
  - One or many file upload body parts and one JSON body part using multipart/mixed
- Content of JSON must be an object
- XML
- Top-level array
- Raw/application/octet-stream
- External REST APIs that are described using OpenAPI 3.0
- Swagger documents of external REST APIs that have metadata regarding content types such as multipart/form-data, multipart/mixed, and application/octet-stream
- REST Adapter invoke connections cannot consume Swagger documents with recurring nested structures. For example:

  ```
  Employee &#8594; payroll &#8594; item (line 1, line2)
  Manager &#8594; payroll  &#8594; item (line 3, line4)
  ```

  In such a case, the Swagger parser caches the first definition of `item` and uses that causing incorrect manager schema. This issue is addressed, but the fix is currently not enabled due to backward compatibility.

  REST endpoints that do not have a request or a response cannot be consumed using a Swagger-based connection or using the local integration. If an error appears after initializing the local integration or the Adapter Endpoint Configuration Wizard with a Swagger-based connection, check the limitations on consumption of certain Swaggers.

Note these additional restrictions:

- The missing server URL for the OpenAPI specification is resolved from the OpenAPI host port. If the server URL is missing from the OpenAPI specification, the base URL is resolved from the domain/host:port of the OpenAPI URL where the OpenAPI specification is hosted. For an example, the target endpoint is resolved using the host and port where the OpenAPI specification is hosted as follows:

  ```
  http(s)://host:port/path-from-openapi
  ```

- Headers/custom headers must be added manually and are not discovered from the OpenAPI specification.
- Connectivity agent limitations:
  - Swagger API consumption does not work with the connectivity agent.
  - OpenAPI consumption does not work with the connectivity agent.

- If a Swagger or OpenAPI specification defines multiple success responses, Oracle Integration uses the responses based upon the following criteria:
  - Oracle Integration first looks for a success response (`200`).
  - If the definition does not contain a `200` response, Oracle Integration looks for a response definition with `default`.
  - If `default` is not defined, Oracle Integration looks for a response with a `201` definition.

  In each category, Oracle Integration first uses the response corresponding to `application/json`. Otherwise, Oracle Integration uses the first response in the list of responses for the status code.

REST Adapter Use Cases

The REST Adapter can be used to implement the following categories of use cases.

- Modernize the Existing Capability
- Shape the API Based on a Client Application's Needs
- Provide a Coarse-Grained API Based on a Client Domain's Needs
- No Application Adapter for an External REST API
- Convert an Unmanaged API into an OAuth2–Protected API

> **Note:**
>
> When you provision a new instance of Oracle Integration, several sample integrations are automatically included. Many of these samples are configured with the REST Adapter. These fully designed samples help you get up and running quickly and show you how easy it is to activate, invoke, and monitor an integration between endpoints. See Running the Sample Integrations of *Using Integrations in Oracle Integration 3*.

Modernize the Existing Capability

There are scenarios in which partners or in-house client applications can consume only REST APIs. The capability is exposed through non-HTTP interfaces such as JDBC. Or the capability is exposed as a SOAP API. For example, status of the orders may reside in an on-premises database that must be retrieved using a SQL query. You can build an integration that retrieves order status and exposes it as a REST API by configuring the REST Adapter connection as a trigger.

Shape the API Based on a Client Application's Needs

There are scenarios in which partners or in-house or channel-specific client applications warrant only a very small subset of information compared to what is exposed by back end data sources. For example, the Get Order SOAP operation exposed by the back end Oracle ERP Cloud application can return several hundred attributes, while the client applications may need less than one-tenth of that. You can build an integration that consumes the SOAP service to retrieve the order details and exposes them as a REST API by configuring the REST Adapter connection as a trigger. The response message for this new REST API can reflect only the needed set of attributes by the client applications. The mapping of data from the back end

SOAP service to the REST API-specific response message is performed only for the subset of attributes.

Provide a Coarse-Grained API Based on a Client Domain's Needs

There are scenarios in which the partners or in-house or channel-specific client applications warrant an API that may not be exposed at the same level of granularity by back end systems. For example, you want to expose an API to your partners for creating a sales order in your application. However, the sales order application may need multiple service invocations for creating one order. Exposing a single API for creating an order to partners abstracts the internal implementation details. You can accomplish this by developing an integration that can send multiple service invocations to the back end systems and expose them as a single REST API by configuring the REST Adapter connection as a trigger.

No Application Adapter for an External REST API

Even though Oracle Integration delivers many adapters for facilitating integration with specific applications, there are still several applications/capabilities for which specific adapters are missing. In other situations, an integration can be built to invoke these external REST APIs by configuring the REST Adapter connection as an invoke.

Convert an Unmanaged API into an OAuth2–Protected API

Applications with unprotected APIs or APIs protected using user credentials generally are difficult to expose publicly. While an unprotected API can be misused, an API protected using user credentials requires a higher level of trust with the client. Also, a change in user credentials implies that the client applications also need to update the credentials. You can create an integration that invokes such APIs and exposes them through a REST Adapter connection configured as a trigger, which is protected using OAuth 2.

Workflow to Create and Add a REST Adapter Connection to an Integration

You follow a very simple workflow to create a connection with an adapter and include the connection in an integration in Oracle Integration.

Step	Description	More Information
1	Decide where to work	• Work in a project (see why working with projects is preferred in *Using Integrations in Oracle Integration 3*). • Work outside a project.
2	Create the adapter connections for the applications you want to integrate. The connections can be reused in multiple integrations and are typically created by the administrator.	Create a REST Adapter Connection
3	Create the integration. When you do this, you add trigger and invoke connections to the integration.	Understand Integration Creation and Best Practices and Add the REST Adapter Connection to an Integration
4	Map data between the trigger connection data structure and the invoke connection data structure.	Map Data in *Using Integrations in Oracle Integration 3*

Step	Description	More Information
5	(Optional) Create lookups that map the different values used by those applications to identify the same type of object (such as gender codes or country codes).	Manage Lookups in *Using Integrations in Oracle Integration 3*
6	Activate the integration.	Manage Integrations in *Using Integrations in Oracle Integration 3*
7	Monitor the integration on the dashboard.	Monitor Integrations During Runtime in *Using Integrations in Oracle Integration 3*
8	Track payload fields in messages during runtime.	Assign Business Identifiers for Tracking Fields in Messages and Track Integration Instances in *Using Integrations in Oracle Integration 3*
9	Manage errors at the integration level, connection level, or specific integration instance level.	Manage Errors in *Using Integrations in Oracle Integration 3*

2
REST Adapter Concepts

The following sections describe REST Adapter capabilities in more detail.

Topics:

- Authentication Support
- REST API Support
- OpenAPI Support
- Attachment Support
- Header, Token, Query Parameter, and Array Support
- Swagger Support
- Mapper Connectivity Properties Support
- REST Endpoint Support
- Cross-Origin Resource Sharing (CORS) Support
- Complex Schema Support
- Resource Principal Session Token Support
- JWT Assertion Support for Outbound Invocations
- Data Obfuscation Support

Authentication Support

The following sections describe REST Adapter authentication capabilities in more detail.

Topics:

- Authenticate Requests for Invoking Oracle Integration Flows
- Authentication Types
- Role-Based Connections
- Extensibility Support for Multiple OAuth Providers

OAuth 2.0 is the industry-standard protocol for authorization. See OAuth 2.0.

Authenticate Requests for Invoking Oracle Integration Flows

Integrations support multiple authentication methods suited to different applications and use cases. The adapters used as a trigger connection to stand up the endpoints/listener for a specific integration can support one or multiple authentication methods.

The following sections discuss the use cases, pros and cons, prerequisites, and instructions necessary for sending a request for each of the supported authentication methods.

Topics:

- About Requests to Invoke Integrations
- About OAuth 2.0 Grants
- Use OAuth 2.0 Grants in Identity Domain Environments
- Use OAuth 2.0 Grants in Oracle Identity Cloud Service Environments

See OAuth Grant Types.

About Requests to Invoke Integrations

All integrations using this adapter as a trigger connection are protected by default using HTTP Basic Authentication and OAuth token-based authentication.

You currently can authenticate your requests to invoke integrations in either of the following ways:

- Using HTTP Basic Authentication by sending the credentials of the user (that is, created in your identity domain) through the HTTP authorization header
- Sending an OAuth access token in the header while invoking an Oracle Integration endpoint after acquiring the access token from your identity domain that serves as the OAuth authorization provider

You must have the ServiceUser role in your identity domain to invoke integrations.

Invoke Integration Endpoints Using HTTP Basic Authentication

This authentication method allows the credentials belonging to an Oracle Integration user to send the request to invoke an integration. You must create this user in your identity domain and ensure that the user was granted the role for invoking an integration.

The user can be:

- Human - representing a business user such as a sales representative, technician, or any other person for invoking an integration
- Nonhuman - representing a service integration account used by an external client application for invoking an integration

Even though it's easy to implement the authentication scheme, this is the least secure way to send a request to Oracle Integration for invoking an integration. Also, Oracle Integration doesn't recommend this authentication scheme.

In addition, the customer must ensure the credentials, when reset, are provided to the client application that invokes the integration to ensure a new set of credentials are being used from then on.

Assign appropriate users and groups of users to the various Oracle Integration roles. For standard/production configurations, use the ServiceUser role. See Oracle Integration Roles and Assign Oracle Integration Roles to Groups in *Provisioning and Administering Oracle Integration 3*.

Invoke Integration Endpoints Using OAuth Token-Based Authentication

This authentication scheme allows the external client to acquire a token that is also sent as part of the request sent to invoke an integration.

The most important step for an application in the OAuth flow is how the application receives an access token (and optionally a refresh token). A grant type is the mechanism used to retrieve the token. OAuth defines several different access grant types that represent different authorization mechanisms.

Applications can request an access token to access protected endpoints in different ways, depending on the type of grant type specified in the identity provider. A grant is a credential representing the resource owner's authorization to access a protected resource.

The following sections discuss the various grant types and their pros/cons, along with instructions on how to configure the specific grant type.

About OAuth 2.0 Grants

There are several OAuth 2.0 grant types you can use in Oracle Integration. Review the following information to identify the grant type to use for your use case.

Grant Type	About the Grant Type	Use Cases and Risks
JWT user assertion (recommended)	A user assertion is a user token that contains identity information about the user. The user can either represent a human or a service integration account created for identifying a specific calling application. The user assertion is used directly as an authorization grant to obtain an access token. The client details are provided either as an authentication header in the request or as a client assertion. The user assertion grant is more secure than the resource owner password credentials grant because the user's credentials are never exposed. The user assertion workflow: • Is used with confidential clients. The OAuth clients are trusted to assert a user/ service integration account identity on behalf of the user/ service integration account. • The resource owner's credentials (Oracle Integration user) are never accessible to the client application. It just uses the assertion of the resource owner. • It isn't redirection-based. It takes a request only from the client application to the authorization server. The user is not redirected between interfaces to authorize the request. This user assertion grant works as follows: • The client requests an access token by providing a user assertion. The client details are provided either as an authentication header in the request or as a client assertion. • The OAuth service authenticates the client and, if valid, supplies an access token. The JWT user assertion characteristics are as follows:	This grant is used by applications that want to programmatically invoke integrations without any user intervention. The client application impersonates the user by sending the user assertion to the identity domain while requesting token access. An access token is returned in the user context. The user can either represent a human or a service integration account created for identifying a specific calling application. Oracle Integration recommends the use of this grant for acquiring an OAuth access token by the applications that must programmatically start the integration without any user intervention. **Risks** Carefully use this grant (only with first party/trusted clients) because it allows for trivial impersonation to more highly privileged accounts on services. **Usage** See Prerequisites for JWT User Assertions.

Grant Type	About the Grant Type	Use Cases and Risks
	• Does not require the client to have knowledge of user credentials. • There is no browser-based end user interaction. • A refresh token is allowed. • An access token is in the context of the end user. In this OAuth flow: • A user attempts to access a client application by sending a generated user assertion. • The client application requests an access token, and often a refresh token, by providing a user assertion or a third-party user assertion. • The identity domain authorization server returns the access token to the client application. • The client application uses the access token in an API call to invoke the integration.	

Grant Type	About the Grant Type	Use Cases and Risks
Authorization code	The authorization code grant type is used by web and mobile applications. It differs from most of the other grant types by first requiring the application to launch a browser to begin the integration. At a high level, the integration consists of the following steps: • The application opens a browser to send the user to the OAuth server. • The user sees the authorization prompt and approves the application request. • The user is redirected back to the application with authorization code in the query string. • The application exchanges the authorization code for an access token. The authorization code has the following characteristics: • Does not require the client to have knowledge of user credentials. • Is a browser-based end user interaction. • A refresh token is allowed. • An access token is in the context of the end user. In this OAuth flow: • A user clicks a link in a web server client application to request access to protected resources. • The client application redirects the browser to the identity domain authorization endpoint with a request for an authorization code: `oauth2/v1/authorize` • The identity domain authorization server returns an authorization code to the client application through a browser redirect after the resource owner gives consent. • The client application subsequently exchanges the authorization code for an	This grant is used by the applications such as web portals and mobile applications involving user interactions that may end up invoking the integrations. In this type of use case, the user signing in to the web portal/mobile application explicitly provides the consent by authenticating against Oracle Integration to let their application start the integration. **Usage** See Prerequisites for Authorization Code.

Grant Type	About the Grant Type	Use Cases and Risks
	access token, and often a refresh token. • The identity domain authorization server returns the access token to the client application. • The client application uses the access token in an API call to invoke the integration.	
Client credentials	The client uses its client credentials (or other supported means of authentication) to request an access token when requesting access to protected resources: • Under its control • Those of another resource owner that have been previously arranged with the authorization server Only confidential clients must use this grant type. In this OAuth flow: • The client authenticates with the authorization server and requests an access token from the token endpoint. Because client authentication is used as the authorization grant, no additional authorization request is required. • The authorization server authenticates the client and, if valid, issues an access token. If the request fails client authentication or is invalid, the authorization server returns an error response.	This grant is typically used by clients to obtain an access token outside of the context of a user (for example, to access resources about themselves rather than to access a user's resources). **Usage** See Prerequisites for Client Credentials.

Grant Type	About the Grant Type	Use Cases and Risks
Resource owner password credential (ROPC) (not recommended)	The resource owner's password credentials (that is, the user name and password) can be used by the OAuth client directly as an authorization grant to obtain an access token. The resource owner password credentials grant type is suitable for cases where the resource owner has a trust relationship with the OAuth client. When using the resource owner password credentials grant, the user provides the credentials (user name and password) directly to the application. The application then uses the credentials to obtain an access token from the OAuth token service. The resource owner password credentials grant is a grant workflow where the client application, together with its client identifier and secret, sends the user name and password in exchange for an access token. Instead of the user having to log in and approve the authorization request in a web interface, the user can enter the user name and password in the client application user interface directly. This workflow has different security properties than other OAuth workflows. The primary difference is that the user's password is accessible to the application. This requires a strong trust of the application by the user. The resource owner password credentials grant has the following characteristics: • The client is required to have knowledge of user credentials. • Is not a browser-based end user interaction. • A refresh token is allowed. • An access token is in the context of the end user. In this OAuth flow: • The user clicks a link in the client application requesting access to protected resources. • The client application requests the resource	This grant can be used by applications that want to programmatically invoke the integration without any user intervention. Use this grant only with trusted first-party clients that securely handle user credentials. Even though this grant type can be used by client applications to acquire an OAuth access token to use for sending the request to invoke an integration in a programmatic manner, Oracle Integration does *not* recommend the resource owner password credential grant because of the following risks: **Risks** • This grant type carries a higher risk than other grant types because it maintains the password anti-pattern this protocol seeks to avoid. The client can abuse the password or the password can unintentionally be disclosed to an attacker (for example, through log files or other records kept by the client). • The application can request a scope with complete access to user resources once it possesses the password credential. • Passwords expire. • This grant is currently in a deprecated state. **Usage** See Prerequisites for Resource Owner Password Credentials.

Grant Type	About the Grant Type	Use Cases and Risks
	owner's user name and password. • The user logs in with their user name and password. • The client application exchanges those credentials for an access token, and often a refresh token, from the identity domain authorization server. • The identity domain authorization server returns the access token to the client application. • The client application uses the access token in an API call to invoke the integration.	

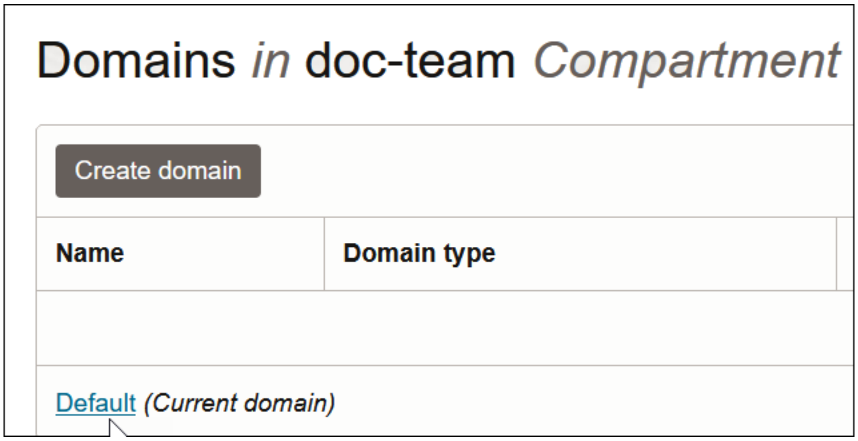
Use OAuth 2.0 Grants in Identity Domain Environments

To use an OAuth 2.0 grant type with this adapter in an identity domain environment of Oracle Integration, you must perform the following prerequisites.

- Access the Identity Domain
- Prerequisites for Client Credentials and Resource Owner Password Credentials
- Prerequisites for JWT User Assertion
- Prerequisites for Authorization Code

Access the Identity Domain

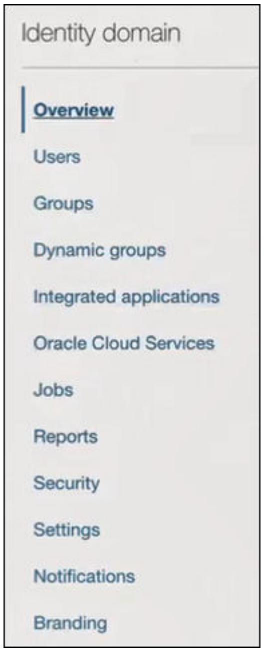
- Log in to the Oracle Cloud Infrastructure Console with your identity domain administrator credentials.
  1. In the navigation pane, click **Identity & Security**.
  2. Click **Domains**.
  3. Select your compartment.
  4. Click the identity domain.



Domains *in* doc-team *Compartment*

Name	Domain type
Default *(Current domain)*	

5. In the navigation pane, click **Integrated applications**.
   This is the location at which you create the client application for your grant type.



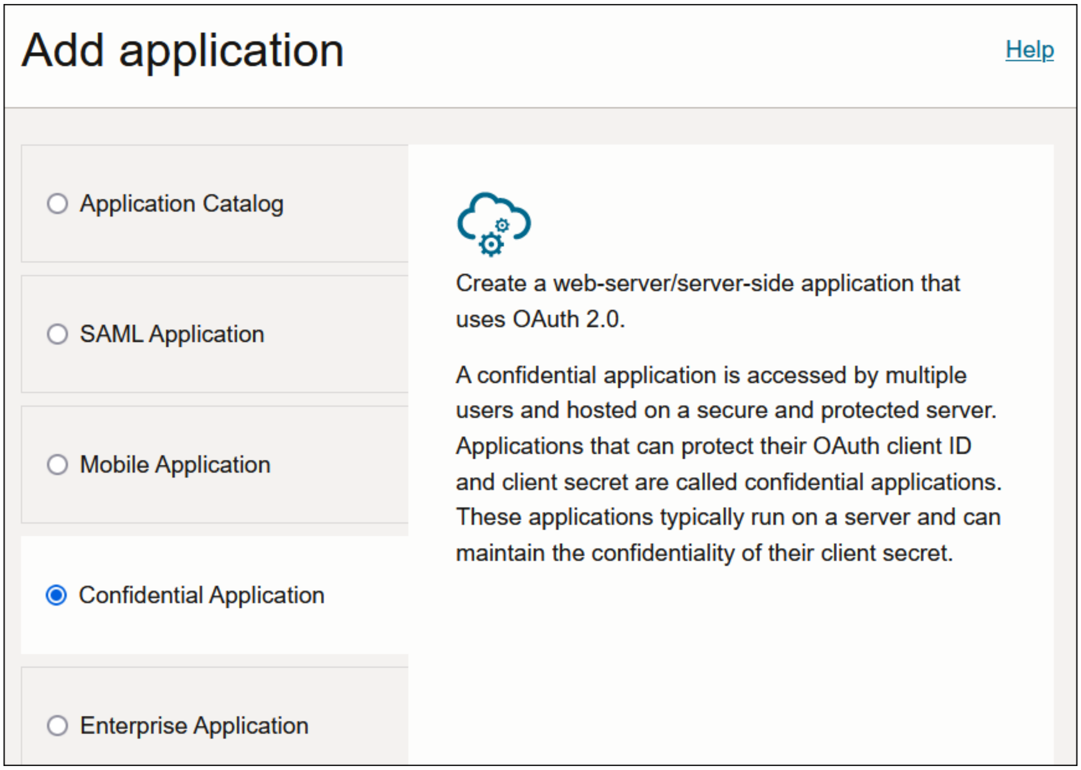
Prerequisites for Client Credentials and Resource Owner Password Credentials

To trigger the integration with OAuth, a client application is required. The prerequisites for the client credentials and resource owner password credentials grant types are very similar.

- Configure the client application
- Add roles to the client application

Configure the client application

1. Click **Add application**.
2. Select **Confidential Application**, then click **Launch workflow**.



Add application

Help

○ Application Catalog

○ SAML Application

○ Mobile Application

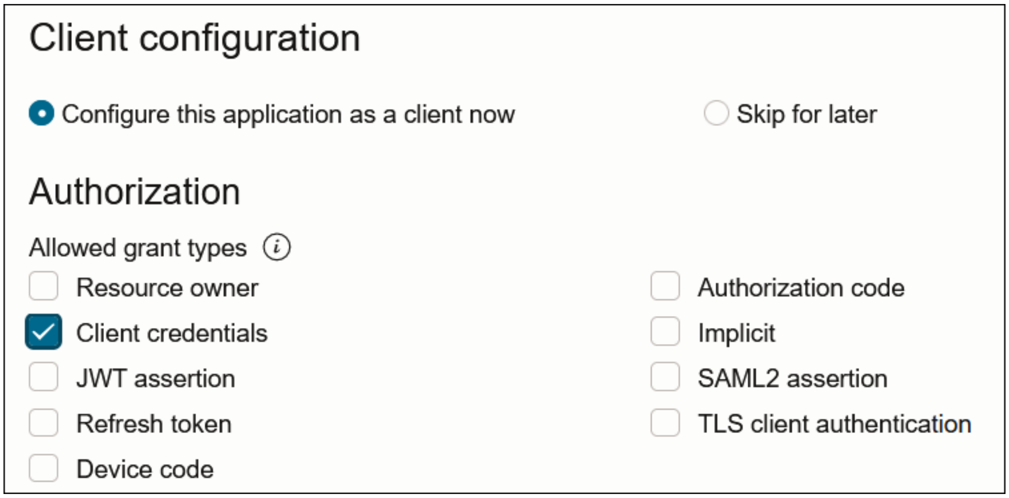
● Confidential Application

○ Enterprise Application

Create a web-server/server-side application that uses OAuth 2.0.

A confidential application is accessed by multiple users and hosted on a secure and protected server. Applications that can protect their OAuth client ID and client secret are called confidential applications. These applications typically run on a server and can maintain the confidentiality of their client secret.

3. Enter a name. The remaining fields on this page are optional and can be ignored.
4. Click **Next**.
5. In the **Client configuration** box, select **Configure this application as a client now**.
6. Select the grant type to use:
    a. For client credentials, select **Client credentials** in the **Allowed grant types** section.



Client configuration

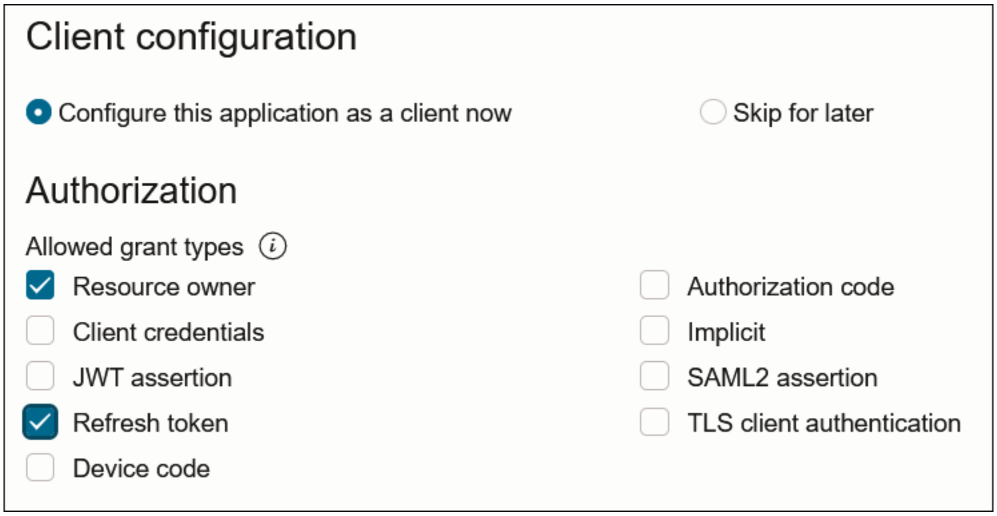
● Configure this application as a client now ○ Skip for later

Authorization

Allowed grant types ⓘ

□ Resource owner	□ Authorization code
☑ Client credentials	□ Implicit
□ JWT assertion	□ SAML2 assertion
□ Refresh token	□ TLS client authentication
□ Device code	

    b. For resource owner password credentials, select **Resource owner** and **Refresh token** in the **Allowed grant types** section.



Client configuration

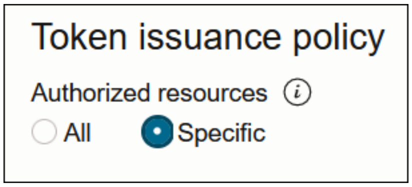
● Configure this application as a client now ○ Skip for later

Authorization

Allowed grant types

☑ Resource owner	□ Authorization code
□ Client credentials	□ Implicit
□ JWT assertion	□ SAML2 assertion
☑ Refresh token	□ TLS client authentication
□ Device code	

7. Complete the following steps for either grant type:
   a. Leave the **Redirect URL**, **Post-logout redirect URL**, and **Logout URL** fields blank.
   b. For **Client type**, ensure that **Confidential** is selected.
   c. Bypass several fields and scroll down to the **Token issuance policy** section.
   d. Select **Specific** in the **Authorized resources** section.

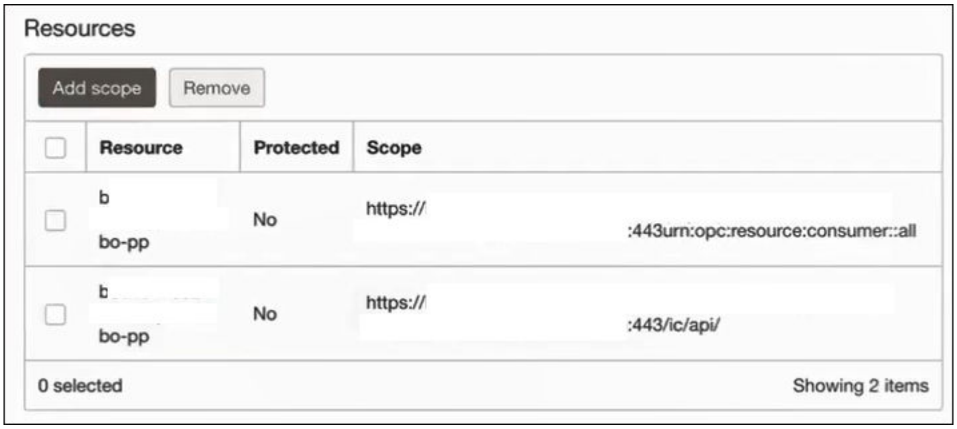


Token issuance policy

Authorized resources

○ All ● Specific

   e. Click the **Add Resources** check box.
   f. Click **Add scope**.
   g. Find the Oracle Integration application for your instance, and click ⌄.
   h. Select the two scopes appended with the following details:
      - **urn:opc:resource:consumer::all**
      - **ic/api/**
   i. Click **Add**.
      The scopes are displayed in the **Resources** section.



Resources

□	Resource	Protected	Scope
□	b bo-pp	No	https:// :443urn:opc:resource:consumer::all
□	b bo-pp	No	https:// :443/ic/api/

0 selected   Showing 2 items

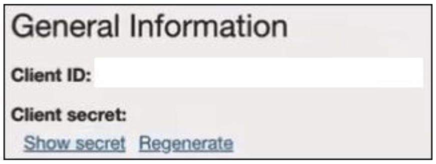
j. Ignore the **Add app roles** check box. This selection is not required.

k. Click **Next**, then click **Finish**.

The details page for the client application is displayed.

8. Click **Activate**, and then **Activate application** to activate the client application for use.

9. In the **General Information** section, note the client ID and client secret values. These values are required for the third-party application that is communicating with the identity domain.



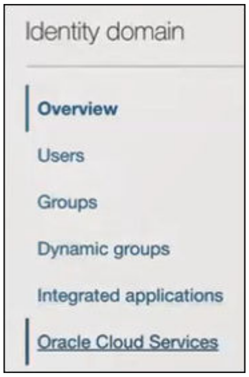
General Information

Client ID:

Client secret:

Add roles to the client application

1. In the navigation pane, click **Oracle Cloud Services**.

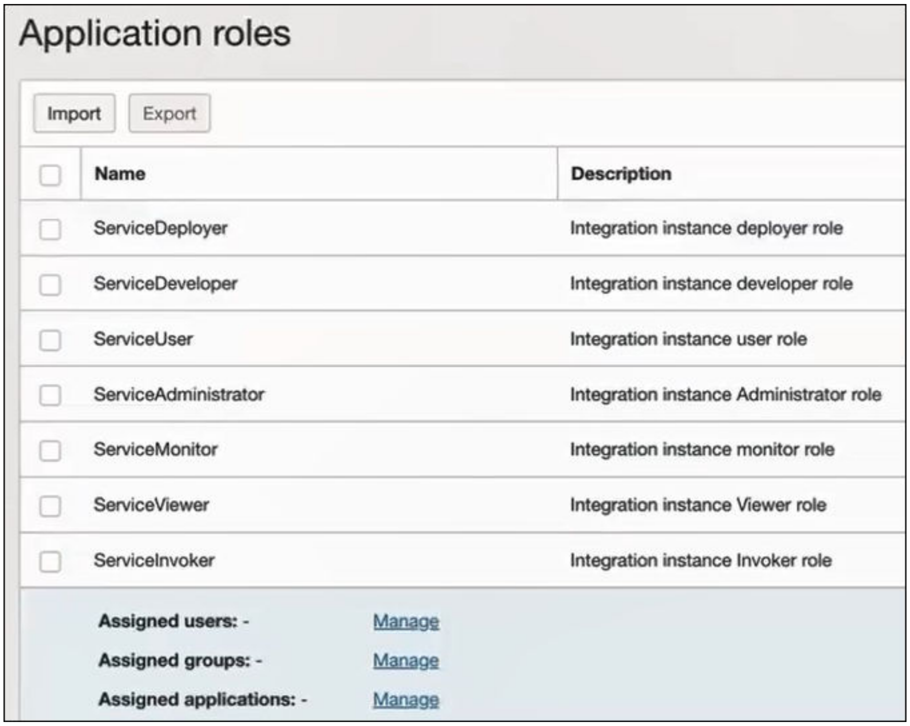


2. Select the specific application corresponding to the Oracle Integration instance.

3. In the navigation pane, click **Application roles**.

4. If configuring the client credentials grant type, select the following:

   a. Expand **ServiceInvoker**, then click **Manage** next to **Assigned applications**.



Application roles

□	Name	Description
□	ServiceDeployer	Integration instance deployer role
□	ServiceDeveloper	Integration instance developer role
□	ServiceUser	Integration instance user role
□	ServiceAdministrator	Integration instance Administrator role
□	ServiceMonitor	Integration instance monitor role
□	ServiceViewer	Integration instance Viewer role
□	ServiceInvoker	Integration instance Invoker role

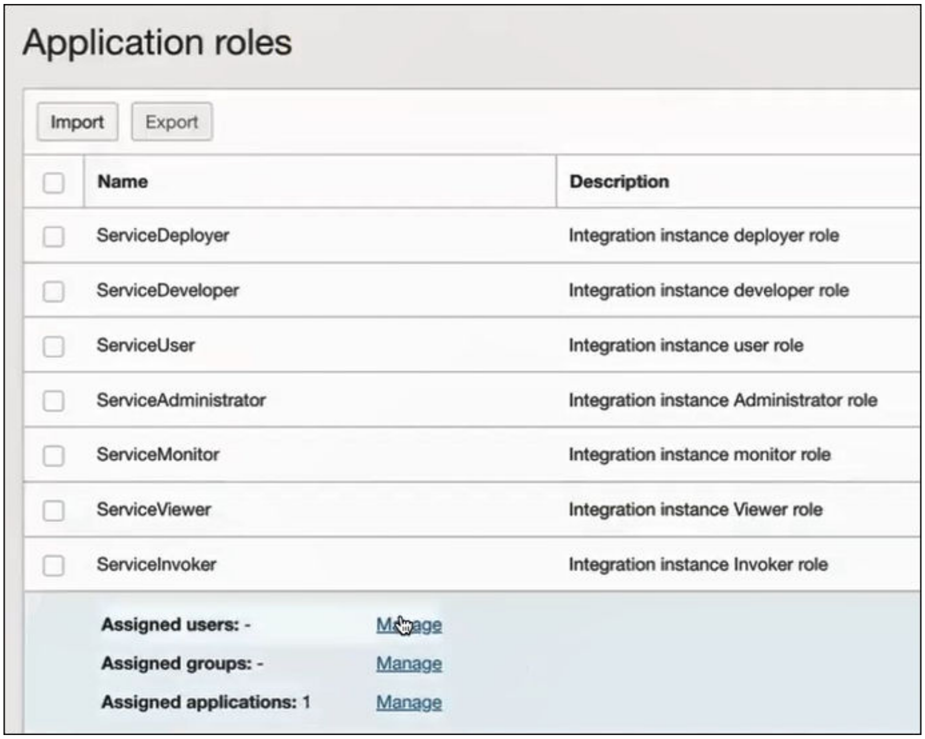
Assigned users: - Manage
Assigned groups: - Manage
Assigned applications: - Manage

b. Click **Show available applications**.

c. Select the application you just created and click **Assign**, then click **Close**.

5. If configuring the resource owner password credentials grant type, select the following:

a. Expand **ServiceInvoker**, then click **Manage** next to either **Assigned users** or **Assigned groups**. For example, if you click **Assigned users**:



Application roles

□	Name	Description
□	ServiceDeployer	Integration instance deployer role
□	ServiceDeveloper	Integration instance developer role
□	ServiceUser	Integration instance user role
□	ServiceAdministrator	Integration instance Administrator role
□	ServiceMonitor	Integration instance monitor role
□	ServiceViewer	Integration instance Viewer role
□	ServiceInvoker	Integration instance Invoker role

Assigned users: - Manage
Assigned groups: - Manage
Assigned applications: 1 Manage

   b. Click **Show available users**.

   c. Select the user and click **Assign**, then click **Close**.

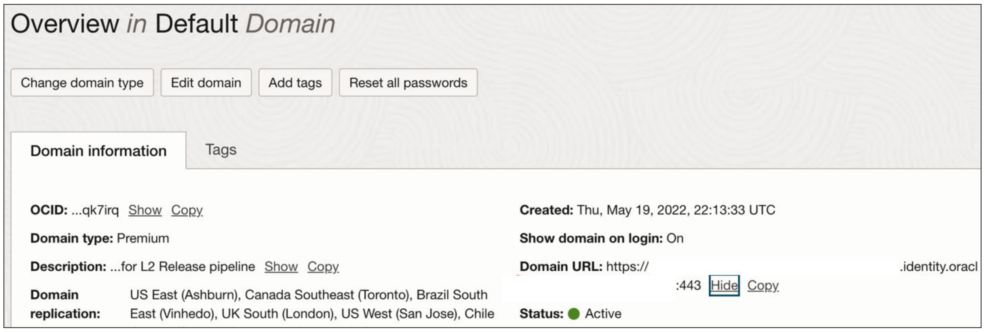
6. Validate the client application for the grant type you are using.

   a. For the client credentials grant type:

      i. Fetch the access client to make an access token request with the client credentials.

```
##Syntax
curl -i -H 'Authorization: Basic <base64Encoded clientid:secret>' -
H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' --
request POST https://
<Identity_Domain_Service_Instance>.identity.oraclecloud.com/
oauth2/v1/token -d 'grant_type=client_credentials&scope=<app scope>'
###where
#### <base64-clientid-secret> - Base 64 encode clientId:ClientSecret
#### <app scope> - Scope added while creating application in client
configuration section (Ends with urn:opc:resource:consumer::all)

##Example
curl -i -H 'Authorization: Basic OGQyM...ZDA0Mjcz' -H 'Content-
Type: application/x-www-form-urlencoded;charset=UTF-8' --request
POST https://<identity_domain_host>/oauth2/v1/token -d
'grant_type=client_credentials&scope=https://<Resource APP
Audience>urn:opc:resource:consumer::all'
```

      Where `Identity_Domain_Service_Instance` is the value in the **Domain URL** field.



Overview *in* Default *Domain*

OCID: ...qk7irq Show Copy

Domain type: Premium

Description: ...for L2 Release pipeline Show Copy

Domain replication: US East (Ashburn), Canada Southeast (Toronto), Brazil South East (Vinhedo), UK South (London), US West (San Jose), Chile

Created: Thu, May 19, 2022, 22:13:33 UTC

Show domain on login: On

Domain URL: https:// .identity.oracl :443 Hide Copy

Status: ● Active

      ii. Capture the `access_token` from the response.

```
{
    "access_token": "eyJ4NXQjG...dfsdfsFgets2ed",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

      iii. Use the `access_token` in the authorization header to invoke the trigger endpoint.

```
curl --location --request GET 'https://OIC host/OIC endpoint' \
--header 'Authorization: Bearer eyJ4NXQjG...dfsdfsFgets2ed'
```

b. For the resource owner password credentials grant type:

   i. To fetch the access client, make a request with the user name and password in the payload.

      ```
      ##Syntax
      curl -i -H 'Authorization: Basic <base64Encoded_clientid:secret>' -
      H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' --
      request POST https://
      <Identity_Domain_Service_Instance>.identity.oraclecloud.com/
      oauth2/v1/token -d 'grant_type=password&username=<user-
      name>&password=<password>&scope=<App_Scope>%20offline_access'

      ###where
      #### <base64-clientid-secret> - Base 64 encode clientId:ClientSecret
      #### <username> - user for token needs to be issued (must be in
      serviceinvoker role).
      #### <password> - password for above user
      #### <app_scope> - Scope added while creating application in client
      configuration section (Ends with urn:opc:resource:consumer::all)
      ##Example
      curl -i -H 'Authorization: Basic OGQyM...ZDA0Mjcz' -H 'Content-
      Type: application/x-www-form-urlencoded;charset=UTF-8' --request
      POST https://<identity_domain_host>/oauth2/v1/token -d
      'grant_type=password&username=sampleUser&password=SamplePassword&sco
      pe=https://
      <Resource_APP_Audience>urn:opc:resource:consumer::all%20offline_acce
      ss'
      ```

   ii. Capture the `access_token` and `refresh_token` from the response.

      ```
      {
          "access_token": "eyJ4NXQjG...dfsdfsFgets2ed",
          "token_type": "Bearer",
          "expires_in": 3600,
          "refresh_token": "AQAgY2MzNjVlOTVhOTRh...vM5S0MkrFSpzc="
      }
      ```

   iii. Use the `access_token` in the authorization header to invoke the Oracle Integration trigger endpoint.

      ```
      curl --location --request GET 'https://OIC host/OIC endpoint' \
      --header 'Authorization: Bearer eyJ4NXQjG...dfsdfsFgets2ed'
      ```

   iv. To update the access token, use the refresh token and make a request.

   v. Capture the `access_token` and `refresh_token` from the response for further use.

      ```
      curl -i -H 'Authorization: Basic <base64-clientid-secret>' -H
      'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' --
      request POST https://
      <Identity_Domain_Service_Instance>.identity.oraclecloud.com/
      oauth2/v1/token  -d
      'grant_type=refresh_token&refresh_token=<refresh_token>'

      ##Example
      ```

```
curl -i -H 'Authorization: Basic OGQyM...ZDA0Mjcz' -H 'Content-
Type: application/x-www-form-urlencoded;charset=UTF-8' --request
POST https://
<Identity_Domain_Service_Instance>.identity.oraclecloud.com/
oauth2/v1/token  -d
'grant_type=refresh_token&refresh_token=AQAgY2MzNjVlOTVhOTRh...vM5S0
MkrFSpzc='
```

Prerequisites for JWT User Assertion

- Generate the key
- Configure the client application
- Add a certificate as a trusted partner
- Generate the JWT user assertion
- Validate the client application

Generate the key

You must first generate the key to import when you configure the client application for the JWT user assertion.

1. Generate the self-signed key pair.

   ```
   keytool -genkey -keyalg RSA -alias <your_alias> -keystore <keystore_file> -
   storepass <password> -validity 365 -keysize 2048

   ##example
   keytool -genkey -keyalg RSA -alias assert -keystore sampleKeystore.jks -
   storepass samplePasswd -validity 365 -keysize 2048
   ```

2. Export the public key for signing the JWT assertion.

   ```
   keytool -exportcert -alias <your_alias> -file <filename> -keystore
   <keystore_file> -storepass <password>

   ##example
   keytool -exportcert -alias assert -file assert.cer -keystore
   sampleKeystore.jks -storepass samplePasswd

   ## This should show a success message e.g. Certificate stored in file
   <assert.cer>
   ```

3. Convert the keystore to P12 format.

   ```
   keytool -importkeystore -srckeystore <filename> -srcstorepass <password> -
   srckeypass <password> -srcalias <your_alias> -destalias <your_alias> -
   destkeystore <destFileName> -deststoretype PKCS12 -deststorepass
   <password> -destkeypass <password>

   ##example
   keytool -importkeystore -srckeystore sampleKeystore.jks -srcstorepass
   samplePasswd -srckeypass samplePasswd -srcalias assert -destalias assert -
   destkeystore assert.p12 -deststoretype PKCS12 -deststorepass samplePasswd -
   destkeypass samplePasswd
   ```

```
## This should show a success message e.g. Importing keystore
sampleKeystore.jks to assert.p12...
```

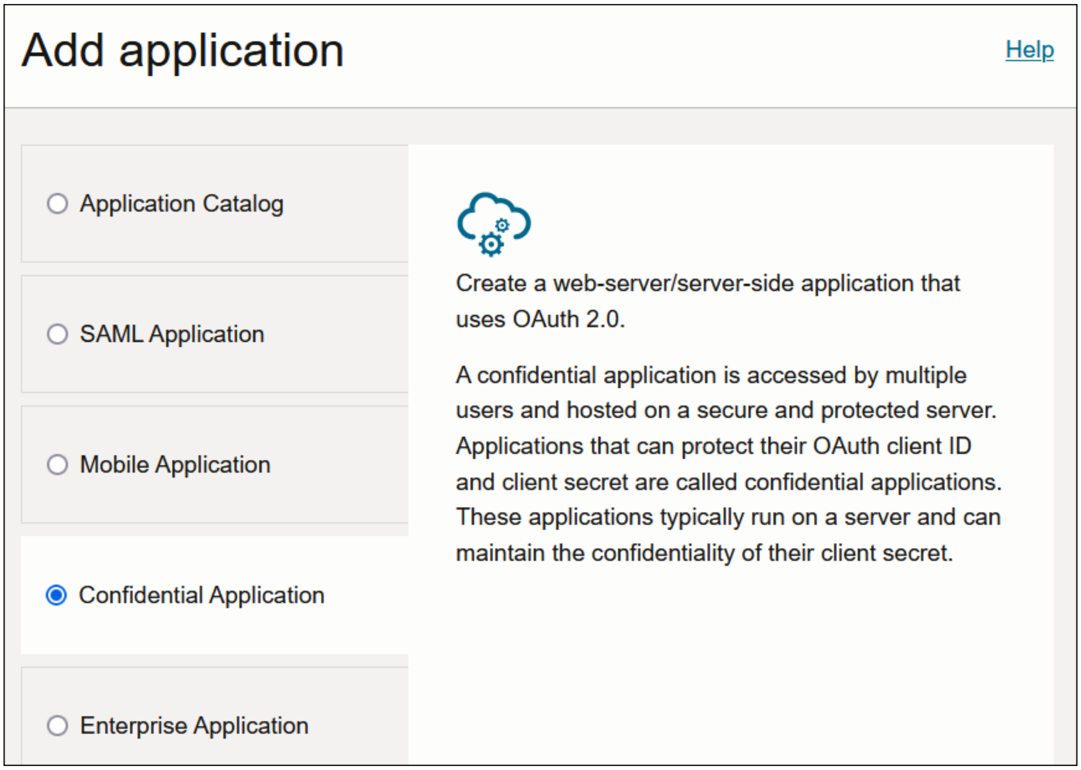
4. Export the private key from the P12 keystore.

   ```
   openssl pkcs12 -in <destFileName> -nodes -nocerts -out <pem_file>

   ##example
   openssl pkcs12 -in assert.p12 -nodes -nocerts -out private_key.pem

   ## This should show a success message: MAC verified OK
   ```

Configure the client application

To trigger the integration with OAuth, a client application is required.

1. Click **Add application**.
2. Select **Confidential Application**, and click **Launch workflow**.



Add application

○ Application Catalog

○ SAML Application

○ Mobile Application

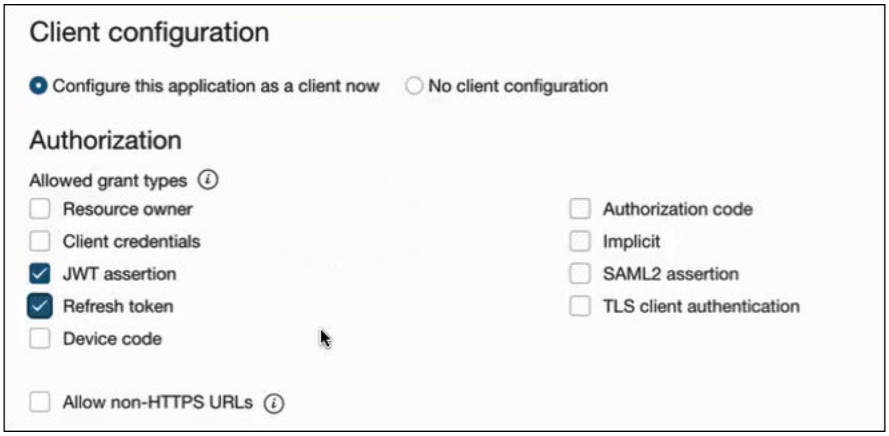
● Confidential Application

○ Enterprise Application

Create a web-server/server-side application that uses OAuth 2.0.

A confidential application is accessed by multiple users and hosted on a secure and protected server. Applications that can protect their OAuth client ID and client secret are called confidential applications. These applications typically run on a server and can maintain the confidentiality of their client secret.

3. Enter a name. The remaining fields on this page are optional and can be ignored.
4. Click **Next**.
5. In the **Client configuration** box, select **Configure this application as a client now**.
6. For JWT user assertions, select **JWT assertion** and **Refresh token** in the **Allowed grant types** section.



Client configuration

● Configure this application as a client now ○ No client configuration

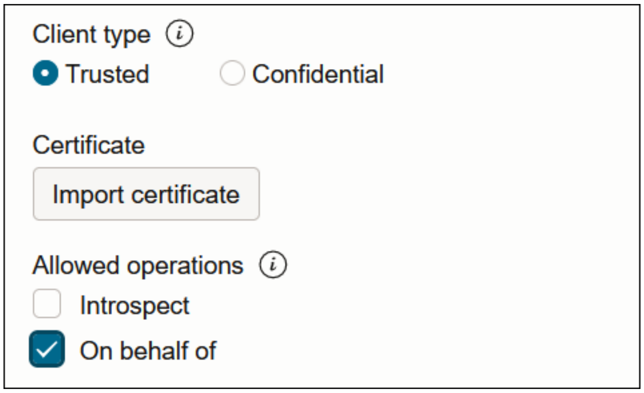
Authorization

Allowed grant types ⓘ

□ Resource owner	□ Authorization code
□ Client credentials	□ Implicit
☑ JWT assertion	□ SAML2 assertion
☑ Refresh token	□ TLS client authentication
□ Device code	

□ Allow non-HTTPS URLs ⓘ

7. Complete the following steps for the grant type:
   a. Leave the **Redirect URL**, **Post-logout redirect URL**, and **Logout URL** fields blank.
   b. In the **Client type** section, select **Trusted**.



Client type ⓘ

● Trusted ○ Confidential

Certificate

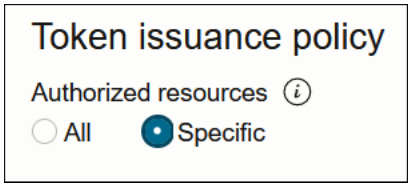
Import certificate

Allowed operations ⓘ

□ Introspect

☑ On behalf of

   c. Upload the certificate created in section Generate the key. This action adds the certificate as a trusted partner.
   d. Bypass several fields and scroll down to the **Token issuance policy** section.
   e. Select **Specific** in the **Authorized resources** section.



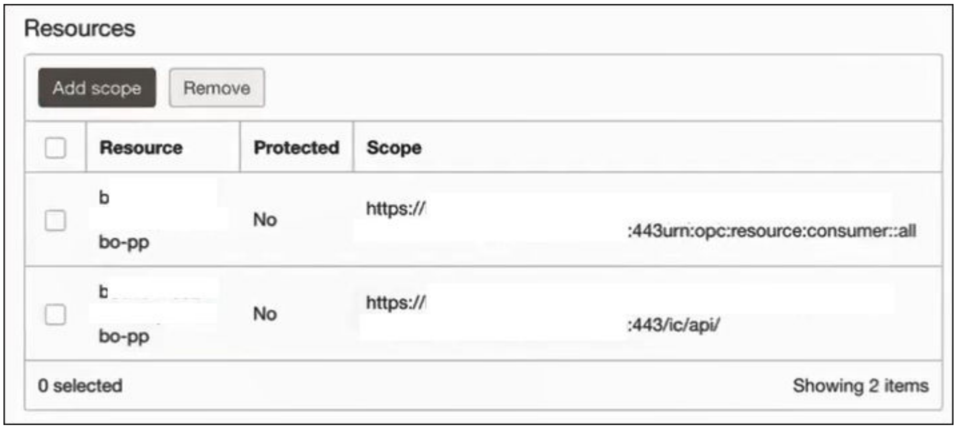
Token issuance policy

Authorized resources ⓘ

○ All ● Specific

   f. Click the **Add Resources** check box.
   g. Click **Add scope**.
   h. Find the Oracle Integration application for your instance, and click ⌄.
   i. Select the two scopes appended with the following details:
      - **urn:opc:resource:consumer::all**
      - **ic/api/**

j. Click **Add**.
The scopes are displayed in the **Resources** section.


Resources

☐	Resource	Protected	Scope
☐	b [redacted] bo-pp	No	https:// [redacted] :443urn:opc:resource:consumer::all
☐	b [redacted] bo-pp	No	https:// [redacted] :443/ic/api/

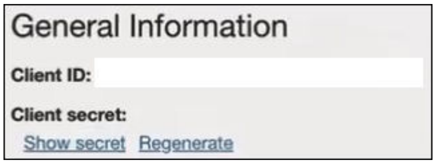
0 selected
Showing 2 items

k. Ignore the **Add app roles** check box. This selection is not required.

l. Click **Next**, then click **Finish**.
The details page for the client application is displayed.

m. Click **Activate**, and then **Activate application** to activate the client application for use.

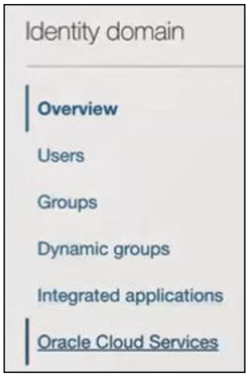
n. In the **General Information** section, note the client ID and client secret values. These values are required for the third-party application that is communicating with the identity domain.


General Information

Client ID: [redacted]

Client secret:

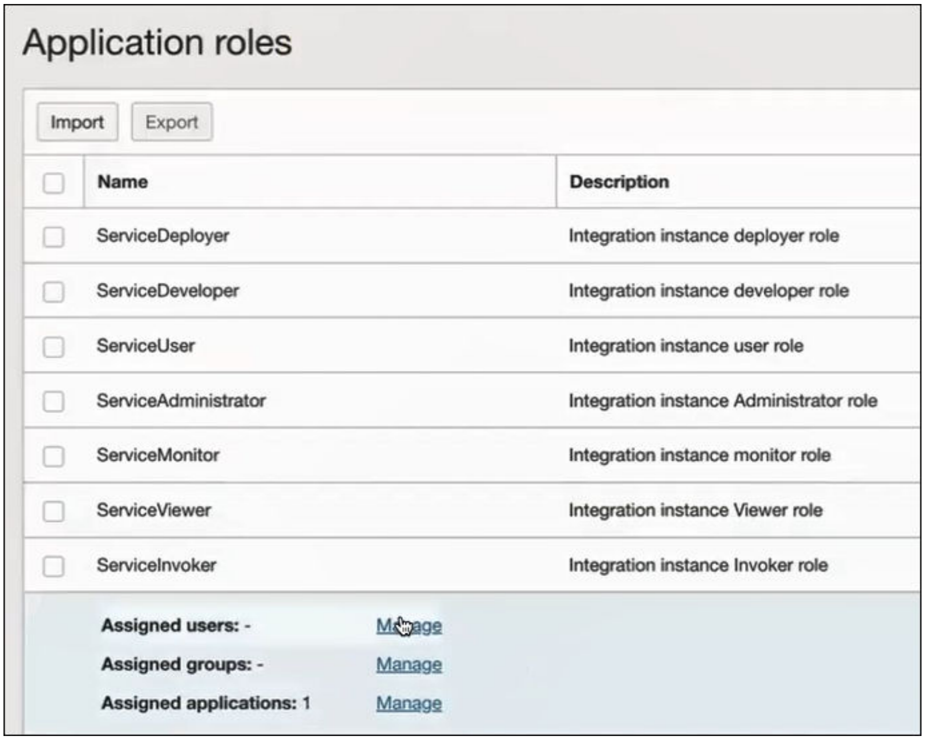
8. In the navigation pane, click **Oracle Cloud Services**.



9. Select the specific application corresponding to the Oracle Integration instance.

10. In the navigation pane, click **Application roles**.

11. Expand **ServiceInvoker**, then click **Manage** next to either **Assigned users** or **Assigned groups**. For example, if you click **Assigned users**:



□	Name	Description
□	ServiceDeployer	Integration instance deployer role
□	ServiceDeveloper	Integration instance developer role
□	ServiceUser	Integration instance user role
□	ServiceAdministrator	Integration instance Administrator role
□	ServiceMonitor	Integration instance monitor role
□	ServiceViewer	Integration instance Viewer role
□	ServiceInvoker	Integration instance Invoker role

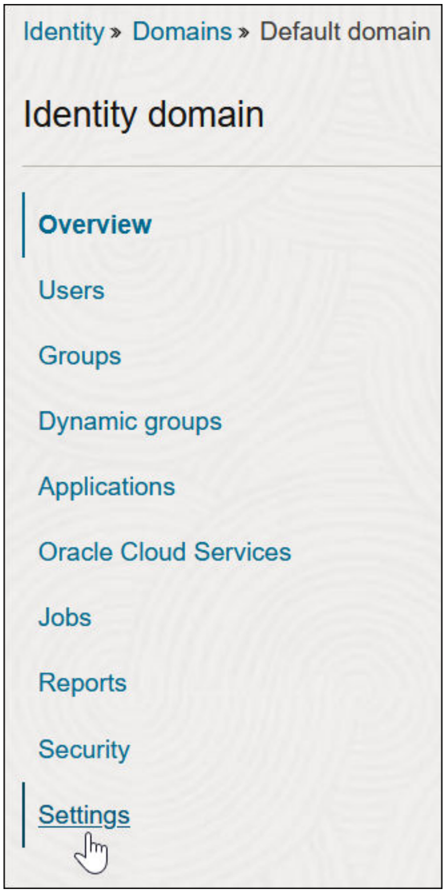
Assigned users: -	Manage
Assigned groups: -	Manage
Assigned applications: 1	Manage

12. Click **Show available users**.
13. Select the user and click **Assign**, then click **Close**.

Add a certificate as a trusted partner

In addition to importing the signing certificate into the client application, you are also required to include the certificate as a trusted partner certificate.

1. In the navigation pane, click **Settings**.



2. Click **Trusted partner certificates**.
3. Click **Import certificate** to upload the certificate created in section Generate the key.

Generate the JWT user assertion

1. Generate the JWT user assertion using the generated private key and simple Java code.

> **Note:**
>
> You can use the https://github.com/jwtk/jjwt library to generate the user assertion. There are many libraries listed at https://jwt.io/ for multiple technologies.

```
Sample:
header:
{
"alg": "RS256",
"typ": "JWT",
"kid": "assert"
}

payload:
{
"sub": "ssaInstanceAdmin",
"jti": "8c7df446-bfae-40be-be09-0ab55c655436",
"iat": 1589889699,
"exp": 1589909699,
"iss": "d702f5b31ee645ecbc49d05983aaee54",
"aud": "https://identity.oraclecloud.com/"
}
```

Where:

- `sub` specifies the user name for whom user assertion is generated.
- `jti` is a unique identifier
- `iat` is issued (epoch seconds).
- `exp` is the token expiry (epoch seconds).
- `iss` is the client ID.
- `aud` must include the identity domain audience `https://identity.oracle.com/`. The signing algorithm must be RS256.
- `kid` specifies the key to use to verify the signature. Therefore, it must match with the uploaded certificate alias.

Validate the client application

1. Once you generate the JWT user assertion, generate the access token as follows.

   ```
   ##Syntax
   curl -i -H 'Authorization: Basic <base64Encoded clientid:secret>' -H
   'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' --request
   POST https://<Identity_Domain_Service_Instance>.identity.oraclecloud.com/
   oauth2/v1/token -d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-
   type%3Ajwt-bearer&assertion=<user assertion>&scope=<app_scope>'

   ###where
   #### grant type - urn:ietf:params:oauth:grant-type:jwt-bearer
   #### <base64-clientid-secret> - Base 64 encode clientId:ClientSecret
   #### <user assertion> - User assertion generated
   #### <app scope> - Scope added while creating application in client
   configuration section (Ends with urn:opc:resource:consumer::all)
   ```

2. Capture the `access_token` from the response.

   ```
   {
       "access_token": "eyJ4NXQjG...dfsdfsFgets2ed",
       "token_type": "Bearer",
       "expires_in": 3600
   }
   ```

3. Use an `access_token` in the authorization header to invoke the Oracle Integration trigger endpoint.

   ```
   curl --location --request GET 'https://OIC host/OIC endpoint' \
   --header 'Authorization: Bearer eyJ4NXQjG...dfsdfsFgets2ed'
   ```

Prerequisites for Authorization Code

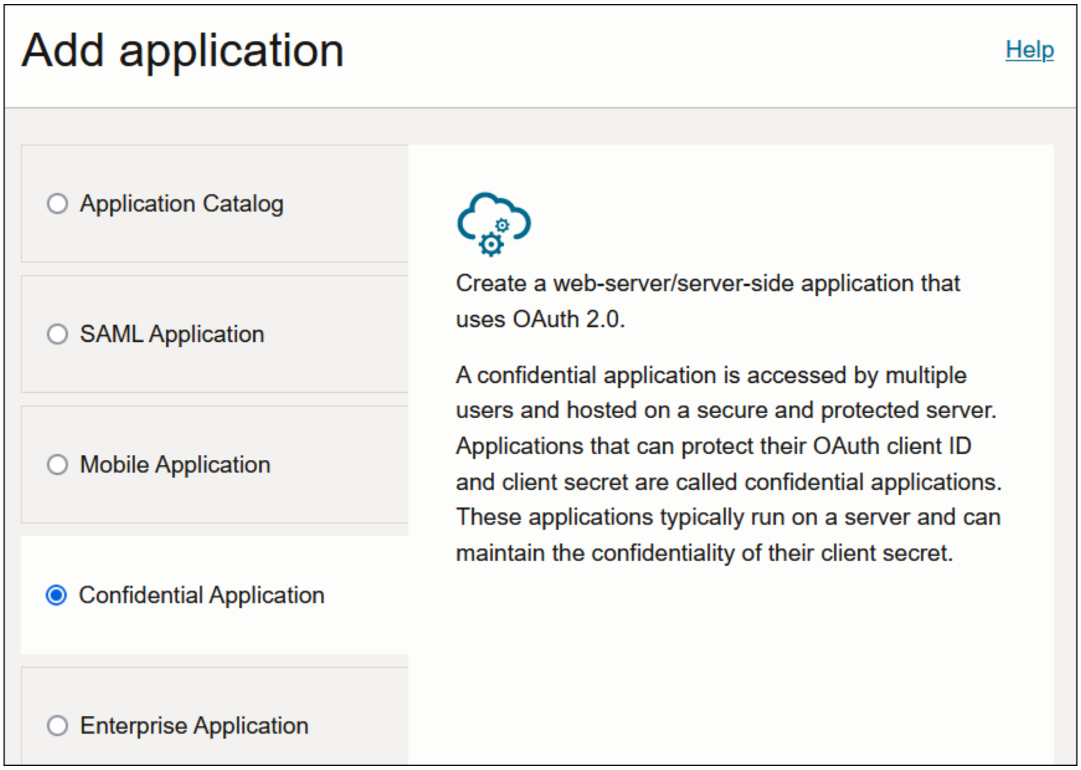
- Configure the client application

- Validate the Oracle Integration application and user roles
- Validate the client application

Configure the client application

To trigger the integration with OAuth, a client application is required.

1. Click **Add application**.
2. Select **Confidential Application**. then click **Launch workflow**.



Add application

Help

○ Application Catalog

○ SAML Application

○ Mobile Application

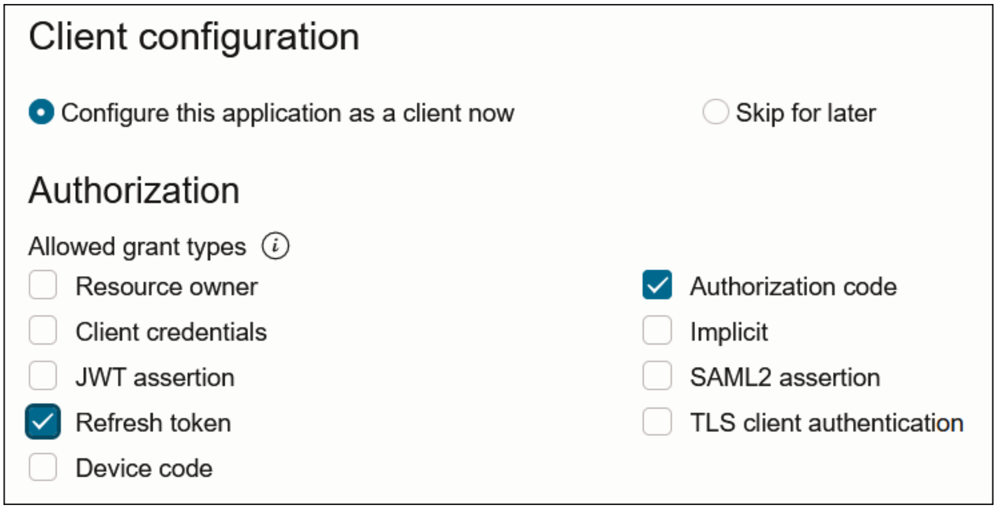
● Confidential Application

○ Enterprise Application

Create a web-server/server-side application that uses OAuth 2.0.

A confidential application is accessed by multiple users and hosted on a secure and protected server. Applications that can protect their OAuth client ID and client secret are called confidential applications. These applications typically run on a server and can maintain the confidentiality of their client secret.

3. Enter a name. The remaining fields on this page are optional and can be ignored.
4. Click **Next**.
5. In the **Client configuration** box, select **Configure this application as a client now**.
6. Select the grant type to use:
   a. For authorization code, select **Refresh token** and **Authorization code** in the **Allowed grant types** section.



Client configuration

● Configure this application as a client now ○ Skip for later

Authorization

Allowed grant types

□ Resource owner	☑ Authorization code
□ Client credentials	□ Implicit
□ JWT assertion	□ SAML2 assertion
☑ Refresh token	□ TLS client authentication
□ Device code	

b. In the **Redirect URL** field, enter the redirect URL of the client application. After user login, this URL is redirected to with the authorization code. You can specify multiple redirect URLs. This is useful for development environments in which you have multiple instances, but only one client application due to licensing issues.

> **Note:**
>
> If you don't know the following information, check with your administrator:
>
> - If your instance is new or upgraded from Oracle Integration Generation 2 to Oracle Integration 3.
> - The complete instance URL with the region included (required for new instances).

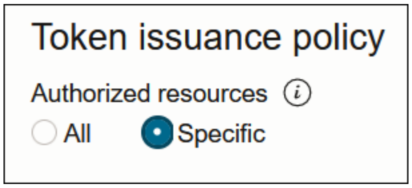
For Connections…	Include the Region as Part of the Redirect URL?	Example of Redirect URL to Specify…

Created on new Oracle Integration 3 instances	Yes.	`https://`
		`OIC_instance_URL.region``.ocp.oraclecloud.com/icsapis/agent/oauth/callback`

Created on instances upgraded from Oracle Integration Generation 2 to Oracle Integration 3	No.
	This applies to both:
	• New connections created after the upgrade
	• Existing connections that were part of the upgrade
		`https://`
		`OIC_instance_URL``.ocp.oraclecloud.com/icsapis/agent/oauth/callback`

c. In the **Client type** section, click **Confidential**.

d. Select **Specific** in the **Authorized resources** section.



Token issuance policy

Authorized resources ⓘ

○ All ● Specific

e. Click the **Add Resources** check box.

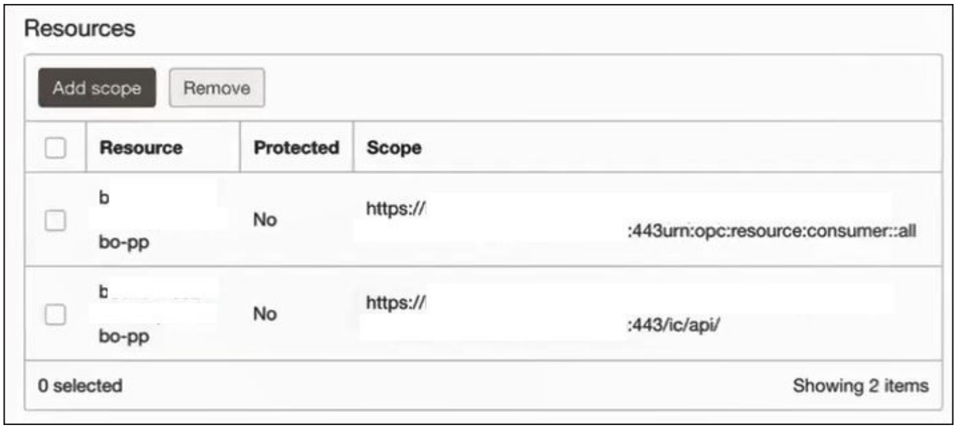
f. Click **Add scope**.

g. Find the Oracle Integration application for your instance, and click ⌄.

h. Select the two scopes appended with the following details:

   - **urn:opc:resource:consumer::all**
   - **ic/api/**

i. Click **Add**.
   The scopes are displayed in the **Resources** section.



Resources

□	Resource	Protected	Scope
□	b bo-pp	No	https:// :443urn:opc:resource:consumer::all
□	b bo-pp	No	https:// :443/ic/api/

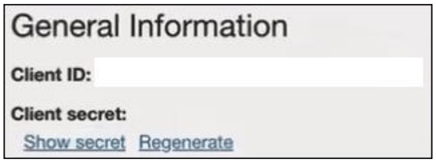
0 selected Showing 2 items

j. Ignore the **Add app roles** check box. This selection is not required.

k. Click **Next**, then click **Finish**.
   The details page for the client application is displayed.

l. Click **Activate**, and then **Activate application** to activate the client application for use.

m. In the **General Information** section, note the client ID and client secret values. These values are required for the third-party application that is communicating with the identity domain.



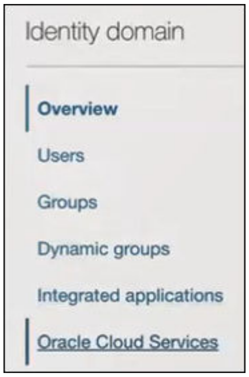
General Information

Client ID:

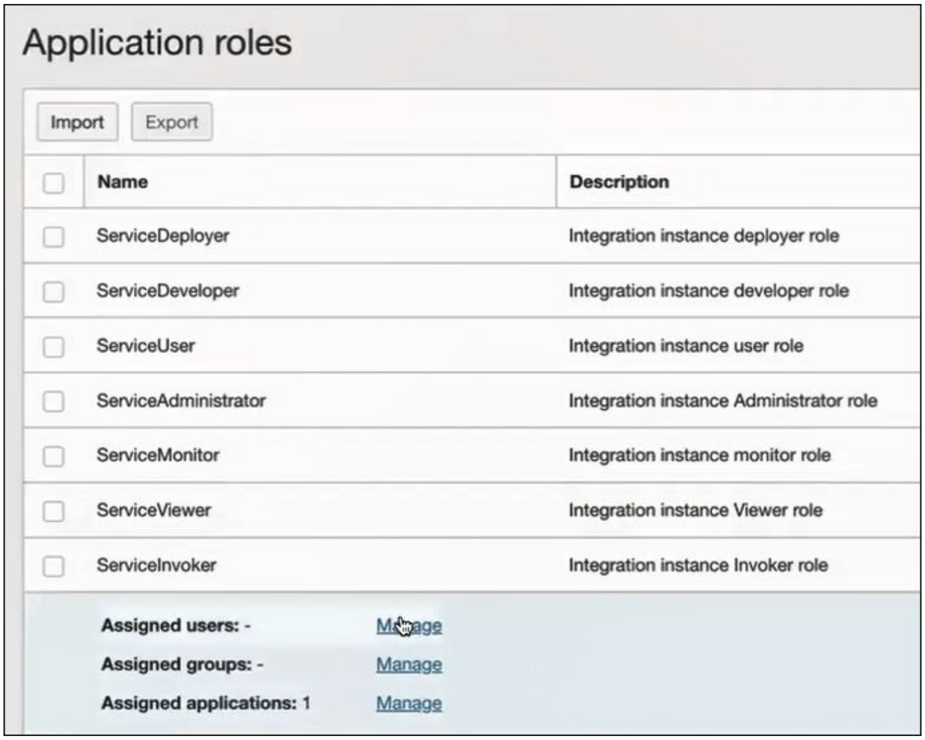
Client secret:

Validate the Oracle Integration application and user roles

1. In the navigation pane, click **Oracle Cloud Services**.



2. Select the specific application corresponding to the Oracle Integration instance.
3. In the navigation pane, click **Application roles**.
4. Expand **ServiceInvoker**, then click **Manage** next to either **Assigned users** or **Assigned groups**. For example, if you click **Assigned users**:



□	Name	Description
□	ServiceDeployer	Integration instance deployer role
□	ServiceDeveloper	Integration instance developer role
□	ServiceUser	Integration instance user role
□	ServiceAdministrator	Integration instance Administrator role
□	ServiceMonitor	Integration instance monitor role
□	ServiceViewer	Integration instance Viewer role
□	ServiceInvoker	Integration instance Invoker role

Assigned users: - Manage
Assigned groups: - Manage
Assigned applications: 1 Manage

5. Click **Show available users**.
6. Select the user and click **Assign**, then click **Close**.

Validate the client application

1. To fetch the authorization code, make the following request from the browser.

```
##Syntax
GET https://<Identity_Domain_Service_Instance>.identity.oraclecloud.com/
oauth2/v1/authorize?client_id=<client-
id>&response_type=code&redirect_uri=<client-redirect-
uri>&scope=<app_scope>%20offline_access&nonce=<nonce-
```

```
value>&state=<unique_value>

###where
#### <client-id> - ID of Client application generated.
#### <client-redirect-uri> - Redirect URI, in client application.
#### <app_scope> - scope added while creating application in client
configuration. (Ends with urn:opc:resource:consumer::all)
#### nonce - Optional, unique value to mitigate replay attacks
#### state - Recommended, Opaque to IDCS. Value used to maintain state
between the request and the callback
##Example
GET https://<identity_domain_host>/oauth2/v1/authorize?
client_id=<clientID>&response_type=code&redirect_uri=https://
app.getpostman.com/oauth2/callback&scope=https://
<Resource_APP_Audience>urn:opc:resource:consumer::all%20offline_access&nonc
e=121&state=12345544
```

2. If the user is not already logged in, you are challenged to authenticate your user credentials. (For authentication, the user assigned the **ServiceInvoker** role must be used.) After authentication is successful, the client URL is redirected to with the authorization code and state added to the URL.

```
##Response URL
https://<redirect_URL>?code=<code_value>=&state=<state_value>

###Client should validate state received is same as one sent in request.
```

3. Capture the code value from the above response and make the following request to get the access token.

```
##Syntax
curl -i -H 'Authorization: Basic <base64-clientid-secret>' -H 'Content-
Type: application/x-www-form-urlencoded;charset=UTF-8' --request POST
https://<Identity_Domain_Service_Instance>.identity.oraclecloud.com/
oauth2/v1/token -d 'grant_type=authorization_code&code=<authz-
code>&redirect_uri=<client-redirect-uri>

###where
#### <base64-clientid-secret> - BAse 64 encode clientId:ClientSecret
#### <authz-code> - code value received as response on redirect.
#### <client-redirect-uri> - Redirect URI, in client application.

##Example
curl -i -H 'Authorization: Basic MDMx..NGY1' -H 'Content-Type:
application/x-www-form-urlencoded;charset=UTF-8' --request POST https://
<identity_domain_host>/oauth2/v1/token -d
'grant_type=authorization_code&code=AQAg...3jKM4Gc=&redirect_uri=https://
app.getpostman.com/oauth2/callback
```

4. Capture the `access_token` and `refresh_token` from the response.

```
{
    "access_token": "eyJ4NXQjG...dfsdfsFgets2ed",
    "token_type": "Bearer",
    "expires_in": 3600,
    "refresh_token": "AQAgY2MzNjVlOTVhOTRh...vM5S0MkrFSpzc="
}
```

5. Use the `access_token` in the authorization header to invoke the Oracle Integration trigger endpoint.

   ```
   curl --location --request GET 'https://OIC host/OIC endpoint' \
   --header 'Authorization: Bearer eyJ4NXQjG...dfsdfsFgets2ed'
   ```

6. To update the access token, use the refresh token and make the request.
7. Capture the `access_token` and `refresh_token` from a response for further use.

   ```
   curl -i -H 'Authorization: Basic <base64-clientid-secret>' -H 'Content-
   Type: application/x-www-form-urlencoded;charset=UTF-8' --request POST
   https://<Identity_Domain_Service_Instance>.identity.oraclecloud.com/
   oauth2/v1/token  -d
   'grant_type=refresh_token&refresh_token=<refresh_token>'


   ##Example
   curl -i -H 'Authorization: Basic OGQyM...ZDA0Mjcz' -H 'Content-Type:
   application/x-www-form-urlencoded;charset=UTF-8' --request POST https://
   <Identity_Domain_Service_Instance>.identity.oraclecloud.com/oauth2/v1/
   token  -d
   'grant_type=refresh_token&refresh_token=AQAgY2MzNjVlOTVhOTRh...vM5S0MkrFSpz
   c='
   ```

Use OAuth 2.0 Grants in Oracle Identity Cloud Service Environments

To use an OAuth 2.0 grant type with this adapter in an Oracle Identity Cloud Service environment of Oracle Integration, you must perform the following prerequisites.

> **Note:**
>
> The following instructions apply *only* to cloud tenancies that still use Oracle Identity Cloud Service. Most cloud tenancies have been migrated to identity domains, which require different configuration instructions. See Use OAuth 2.0 Grants in Identity Domain Environments. If you are unsure of your environment, check with your administrator.

- Prerequisites for All Grants
- Prerequisites for JWT User Assertion
- Prerequisites for Authorization Code
- Prerequisites for Resource Owner Password Credentials
- Prerequisites for Client Credentials

Prerequisites for All Grants

Perform the following tasks for each grant type you use.

- Obtain the Oracle Identity Cloud Service URL.

   1. Go to the URL for your Oracle Integration instance.
      For example, if your Oracle Integration instance is `https://myhost.example.com/ic/home`, when you go to that URL, you are redirected to a URL such as:

      ```
      https://idcs-c2881.identity.myhost.example.com/ui/v1/signin
      ```

   2. Replace `/signin` with `/adminconsole` to access Oracle Identity Cloud Service.
      For example:

      ```
      https://idcs-c2881.identity.myhost.example.com/ui/v1/adminconsole
      ```

      You'll be prompted to sign in again to the Oracle Identity Cloud Service Console.

   3. Log in to the Oracle Identity Cloud Service Console with your identity domain administrator credentials.

- Check the Oracle Integration application in Oracle Identity Cloud Service.
  When an Oracle Integration instance is provisioned, an Oracle Identity Cloud Service application is created for that Oracle Integration instance. The application name is `OICINST_`*`service_instance_name`*.

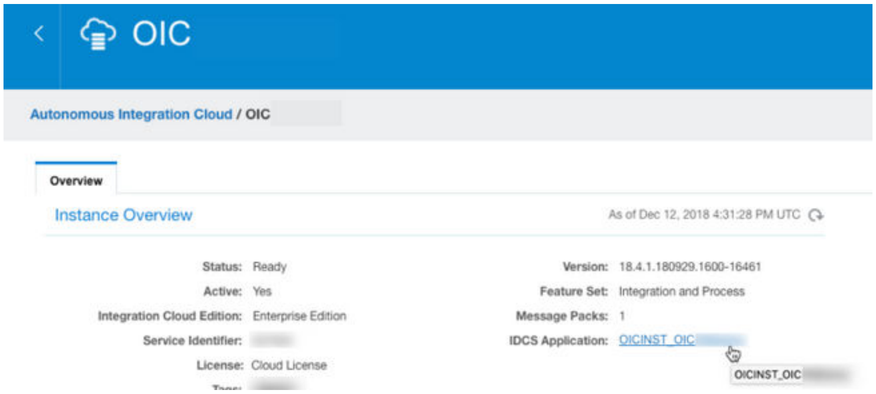
   1. Log in to the Oracle instance to get the service instance name.

      ```
      https://myhost.example.com/ic/home
      ```

   2. Log in to Oracle Identity Cloud Service to get the application.

   3. Go to **Applications** and find the application with the above name to access the application.

  Alternatively, you can find the application through the Oracle Cloud Dashboard. When you click the **IDCS Application** link on the details page of the Oracle Integration instance (for this example, named **OIC**), it opens the Oracle Identity Cloud Service application for Oracle Integration that is already created.



OIC

Autonomous Integration Cloud / OIC

Overview

Instance Overview

As of Dec 12, 2018 4:31:28 PM UTC

Status: Ready
Active: Yes
Integration Cloud Edition: Enterprise Edition
Service Identifier:
License: Cloud License
Tags:

Version: 18.4.1.180929.1600-16461
Feature Set: Integration and Process
Message Packs: 1
IDCS Application: OICINST_OIC

OICINST_OIC

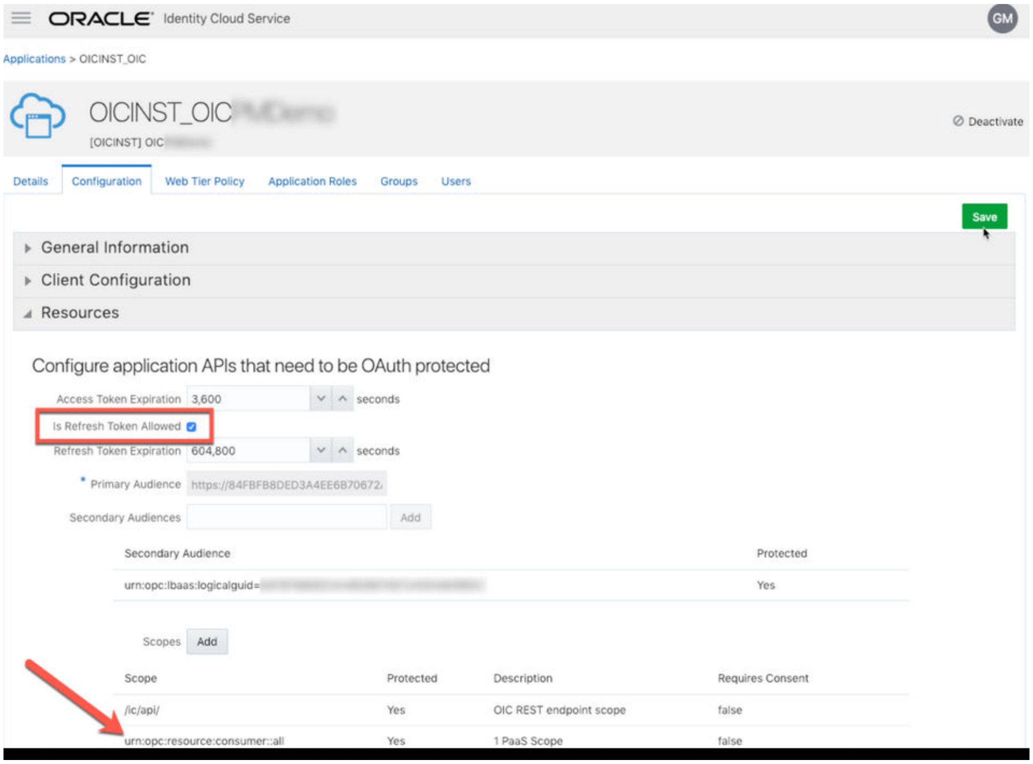
Prerequisites for JWT User Assertion

Perform the following tasks.

- Validate the Oracle Integration application and user roles.

   1. Verify that the **Is Refresh Token Allowed** option is enabled for the Oracle Identity Cloud Service application.

2. Check under the **Configuration** > **Resources** section of the application. Note also that there is a special scope predefined (**urn:opc:resource:consumer::all**), which can trigger integrations using OAuth.



OICINST_OIC[redacted]
[OICINST] OIC[redacted]

Configure application APIs that need to be OAuth protected

Access Token Expiration: 3,600 seconds
Is Refresh Token Allowed ☑
Refresh Token Expiration: 604,800 seconds
* Primary Audience: https://84FBFB8DED3A4EE6B70672/
Secondary Audiences: Add

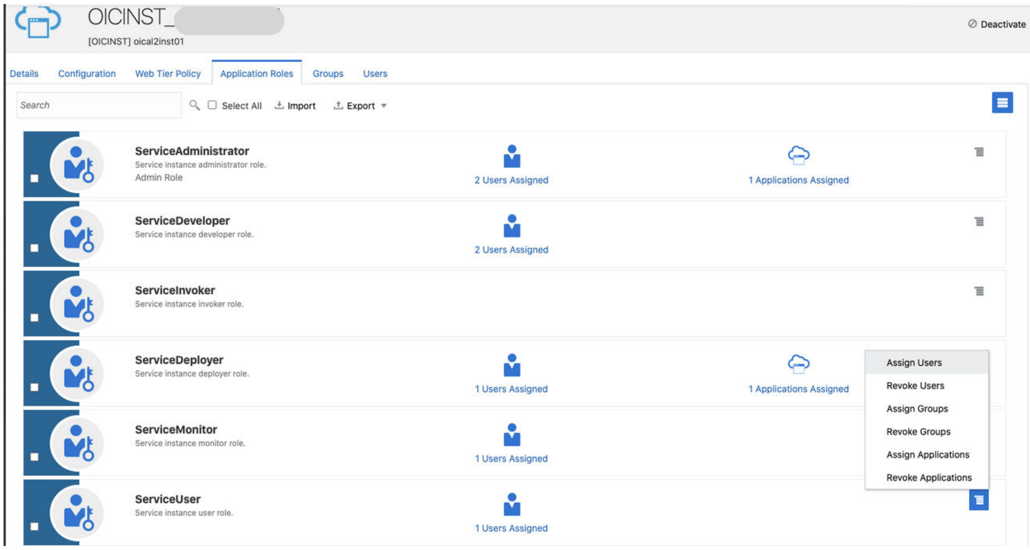
Secondary Audience	Protected
urn:opc:lbaas:logicalguid=[redacted]	Yes

Scopes: Add

Scope	Protected	Description	Requires Consent
/ic/api/	Yes	OIC REST endpoint scope	false
urn:opc:resource:consumer::all	Yes	1 PaaS Scope	false

3. Add the appropriate users to the various Oracle Integration roles. For standard/production configurations, use the **ServiceUser** role. (See Oracle Integration Service Roles in *Provisioning and Administering Oracle Integration 3*.)

4. To assign the user, go to the **Application Roles** section of the application.



OICINST_[redacted]
[OICINST] oical2inst01

Role	Users	Applications
ServiceAdministrator Service instance administrator role. Admin Role	2 Users Assigned	1 Applications Assigned
ServiceDeveloper Service instance developer role.	2 Users Assigned	
ServiceInvoker Service instance invoker role.		
ServiceDeployer Service instance deployer role.	1 Users Assigned	1 Applications Assigned
ServiceMonitor Service instance monitor role.	1 Users Assigned	
ServiceUser Service instance user role.	1 Users Assigned	

- Generate the key:

1. Generate the self-signed key pair.

   ```
   keytool -genkey -keyalg RSA -alias <your_alias> -keystore
   <keystore_file> -storepass <password> -validity 365 -keysize 2048

   ##example
   keytool -genkey -keyalg RSA -alias assert -keystore sampleKeystore.jks -
   storepass samplePasswd -validity 365 -keysize 2048
   ```

2. Export the public key for signing the JWT assertion.

   ```
   keytool -exportcert -alias <your_alias> -file <filename> -keystore
   <keystore_file> -storepass <password>

   ##example
   keytool -exportcert -alias assert -file assert.cer -keystore
   sampleKeystore.jks -storepass samplePasswd

   ## This should show a success message e.g. Certificate stored in file
   <assert.cer>
   ```

3. Convert the keystore to P12 format.

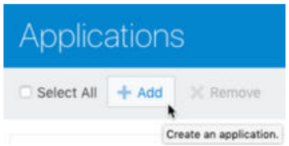
   ```
   keytool -importkeystore -srckeystore <filename> -srcstorepass
   <password> -srckeypass <password> -srcalias <your_alias> -destalias
   <your_alias> -destkeystore <destFileName> -deststoretype PKCS12 -
   deststorepass <password> -destkeypass <password>

   ##example
   keytool -importkeystore -srckeystore sampleKeystore.jks -srcstorepass
   samplePasswd -srckeypass samplePasswd -srcalias assert -destalias
   assert -destkeystore assert.p12 -deststoretype PKCS12 -deststorepass
   samplePasswd -destkeypass samplePasswd

   ## This should show a success message e.g. Importing keystore
   sampleKeystore.jks to assert.p12...
   ```

4. Export the private key from the P12 keystore.

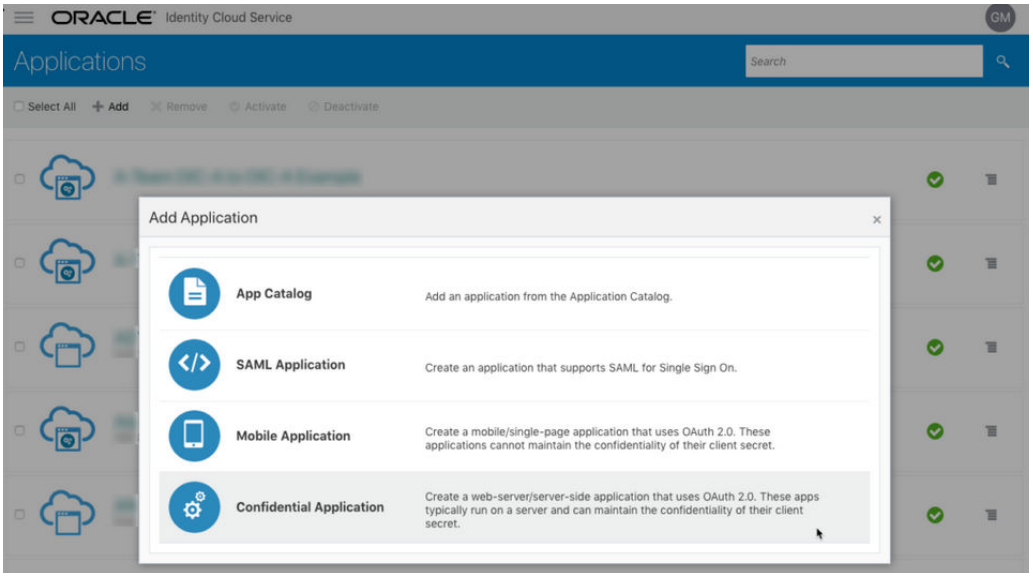
   ```
   openssl pkcs12 -in <destFileName> -nodes -nocerts -out <pem_file>

   ##example
   openssl pkcs12 -in assert.p12 -nodes -nocerts -out private_key.pem

   ## This should show a success message: MAC verified OK
   ```

- Configure the client application:
  To trigger the integration with OAuth, a client application is required.

  1. In the Oracle Identity Cloud Service Console, go to the **Applications** section to create a new application that allows you to trigger an integration with OAuth.



2. Click **Add**.
3. Select **Confidential Application**.



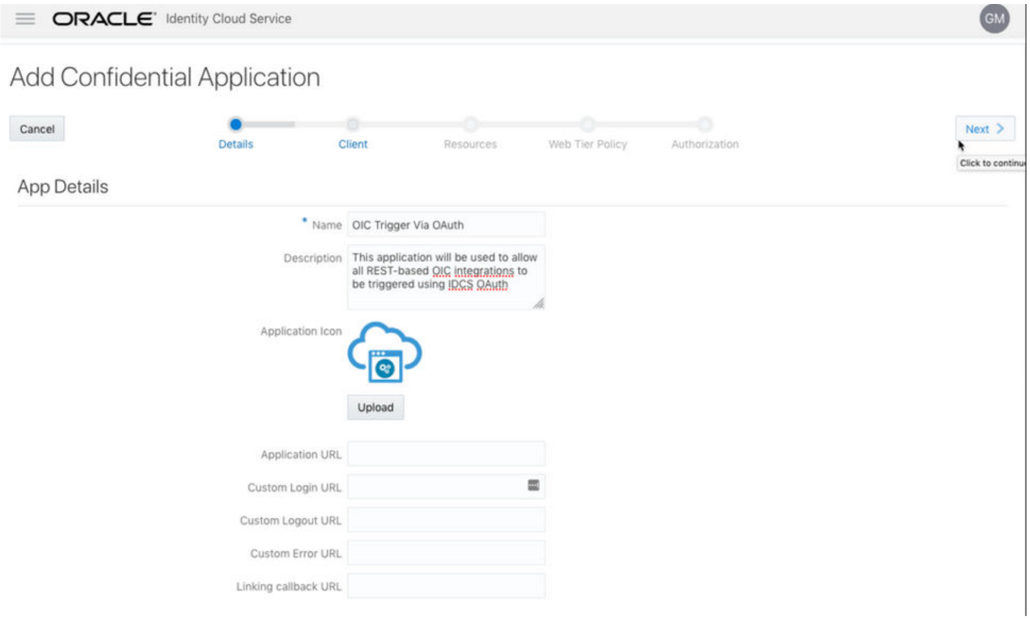
App Catalog — Add an application from the Application Catalog.

SAML Application — Create an application that supports SAML for Single Sign On.

Mobile Application — Create a mobile/single-page application that uses OAuth 2.0. These applications cannot maintain the confidentiality of their client secret.

Confidential Application — Create a web-server/server-side application that uses OAuth 2.0. These apps typically run on a server and can maintain the confidentiality of their client secret.

4. Complete the **Details** page, and go to the **Client** page.



* Name: OIC Trigger Via OAuth

Description: This application will be used to allow all REST-based OIC integrations to be triggered using IDCS OAuth

Application URL

Custom Login URL

Custom Logout URL

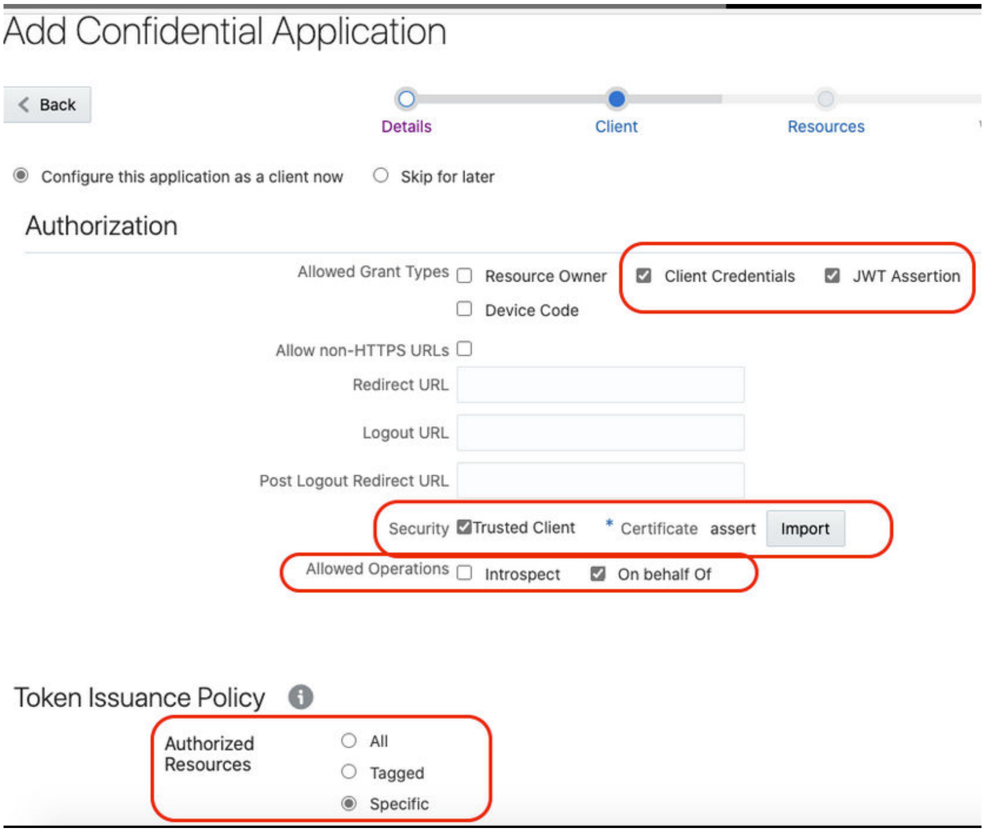
Custom Error URL

Linking callback URL

5. On the **Client** page, select **Configure this application as a client now** and add the following.
   a. Select **Client Credentials** and **JWT Assertion** for the **Allowed Grant Types**.

b. In the **Security** section, select **Trusted Client** and upload the certificate created in the previous section (Generate the key - Step 2).

c. Select **Specific** in the **Authorized Resources** section.



Add Confidential Application

Back

Details Client Resources

● Configure this application as a client now ○ Skip for later

Authorization

Allowed Grant Types □ Resource Owner ☑ Client Credentials ☑ JWT Assertion

□ Device Code

Allow non-HTTPS URLs □

Redirect URL

Logout URL

Post Logout Redirect URL

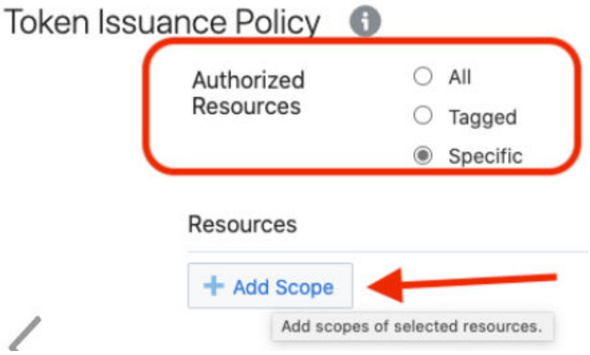
Security ☑Trusted Client * Certificate assert Import

Allowed Operations □ Introspect ☑ On behalf Of

Token Issuance Policy

Authorized Resources ○ All ○ Tagged ● Specific

d. Click **Add Scope** under the **Resources** section.



Token Issuance Policy

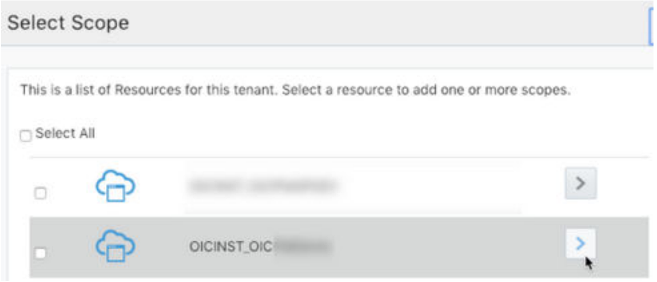
Authorized Resources ○ All ○ Tagged ● Specific

Resources

+ Add Scope

Add scopes of selected resources.

e. Find the Oracle Integration application, and click **>**.



Select Scope

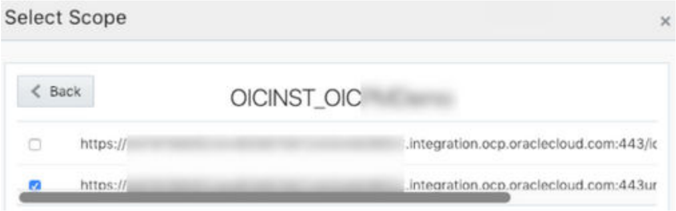
This is a list of Resources for this tenant. Select a resource to add one or more scopes.

□ Select All

□ [redacted]

□ OICINST_OIC[redacted]

f. Add the scope containing **urn:opc:resource:consumer::all**, and click **>**.



Select Scope ×

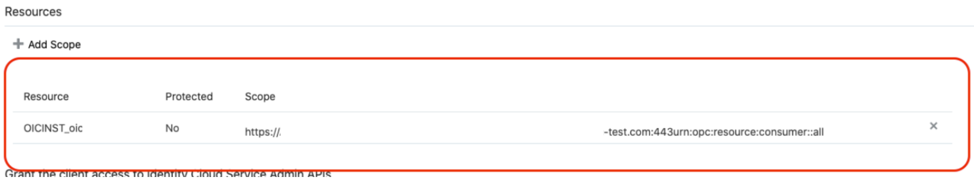
< Back

OICINST_OIC[redacted]

□ https://[redacted].integration.ocp.oraclecloud.com:443/ic

☑ https://[redacted].integration.ocp.oraclecloud.com:443ur

The scope containing **urn:opc:resource:consumer::all** is added.



Resources

+ Add Scope

Resource	Protected	Scope
OICINST_oic	No	https://. -test.com:443urn:opc:resource:consumer::all

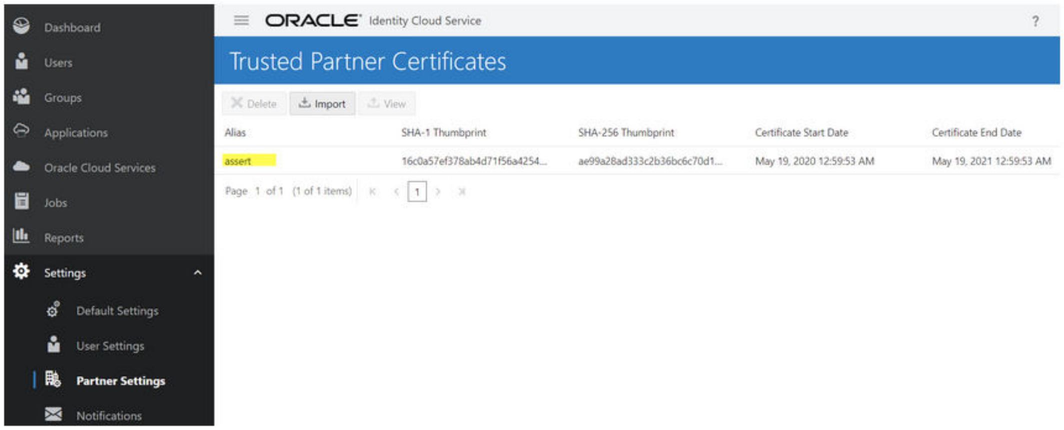
Grant the client access to Identity Cloud Service Admin APIs

g. Save your changes.

6. Click through the remaining wizard pages without making changes and save the application.

7. Activate the application for use.

- Add a certificate as a trusted partner:
  Even though you imported the signing certificate in the application, Oracle Identity Cloud Service requires you to also have the certificate as a trusted partner certificate. Upload the certificate created in the previous section. (See Generate the key - Step 2.)



Trusted Partner Certificates

Alias	SHA-1 Thumbprint	SHA-256 Thumbprint	Certificate Start Date	Certificate End Date
assert	16c0a57ef378ab4d71f56a4254...	ae99a28ad333c2b36bc6c70d1...	May 19, 2020 12:59:53 AM	May 19, 2021 12:59:53 AM

Page 1 of 1 (1 of 1 items) 1

- Generate the JWT user assertion:

  1. Generate the JWT user assertion using the generated private key and simple Java code.

     > **Note:**
     >
     > You can use the https://github.com/jwtk/jjwt library to generate the user assertion. There are many libraries listed at https://jwt.io/ for multiple technologies.

     ```
     Sample:
     header:
     {
     "alg": "RS256",
     "typ": "JWT",
     "kid": "assert"
     }

     payload:
     {
     "sub": "ssaInstanceAdmin",
     "jti": "8c7df446-bfae-40be-be09-0ab55c655436",
     "iat": 1589889699,
     "exp": 1589909699,
     "iss": "d702f5b31ee645ecbc49d05983aaee54",
     "aud": "https://identity.oraclecloud.com/"
     }
     ```

  Where:

  - `sub` specifies the user name for whom user assertion is generated.
  - `jti` is a unique identifier
  - `iat` is issued (epoch seconds).
  - `exp` is the token expiry (epoch seconds).
  - `iss` is the client ID.
  - `aud` must include the Oracle Identity Cloud Service audience `https://identity.oracle.com/`. The signing algorithm must be RS256.
  - `kid` specifies the key to use to verify the signature. Therefore, it must match with the uploaded certificate alias in Oracle Identity Cloud Service.

- Validate the client application:

  1. Once you generate the JWT user assertion, generate the Oracle Identity Cloud Service access token as follows.

     ```
     ##Syntax
     curl -i -H 'Authorization: Basic <base64Encoded clientid:secret>' -H
     'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' --
     request POST https://<IDCS-Service-Instance>.identity.oraclecloud.com/
     oauth2/v1/token -d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-
     type%3Ajwt-bearer&assertion=<user assertion>&scope=<app_scope>'
     ```

```
###where
#### grant type - urn:ietf:params:oauth:grant-type:jwt-bearer
#### <base64-clientid-secret> - Base 64 encode clientId:ClientSecret
#### <user assertion> - User assertion generated
#### <app scope> - Scope added while creating application in client
configuration section (Ends with urn:opc:resource:consumer::all)
```

2. Capture the `access_token` from the response.

   ```
   {
       "access_token": "eyJ4NXQjG...dfsdfsFgets2ed",
       "token_type": "Bearer",
       "expires_in": 3600
   }
   ```

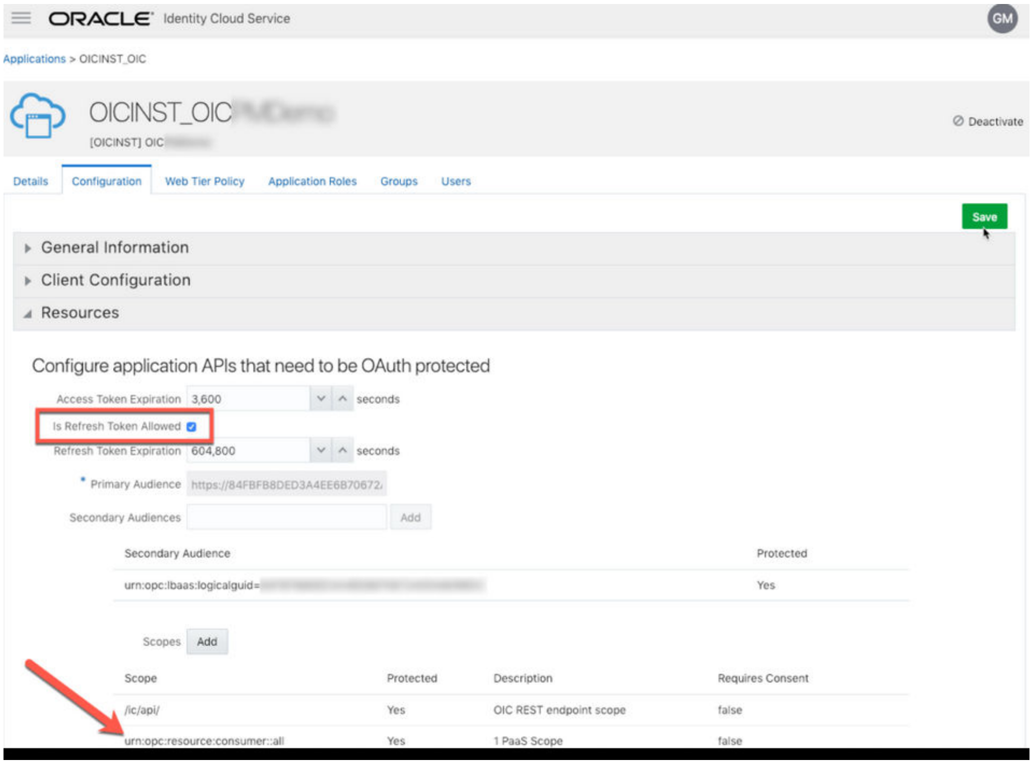
3. Use an `access_token` in the authorization header to invoke the Oracle Integration trigger endpoint.

   ```
   curl --location --request GET 'https://OIC host/OIC endpoint' \
   --header 'Authorization: Bearer eyJ4NXQjG...dfsdfsFgets2ed'
   ```

Prerequisites for Authorization Code

Perform the following tasks.

- Validate the Oracle Integration application and user roles:
  1. Verify that the **Is Refresh Token Allowed** option is enabled for the Oracle Identity Cloud Service application.
  2. Check the **Configuration** > **Resources** section of the application. Note also that there is a special predefined scope (**urn:opc:resource:consumer::all**) that permits triggering of the Oracle Integration integrations using OAuth.



Configure application APIs that need to be OAuth protected

Access Token Expiration 3,600 seconds

Is Refresh Token Allowed ☑

Refresh Token Expiration 604,800 seconds

* Primary Audience https://84FBFB8DED3A4EE6B70672

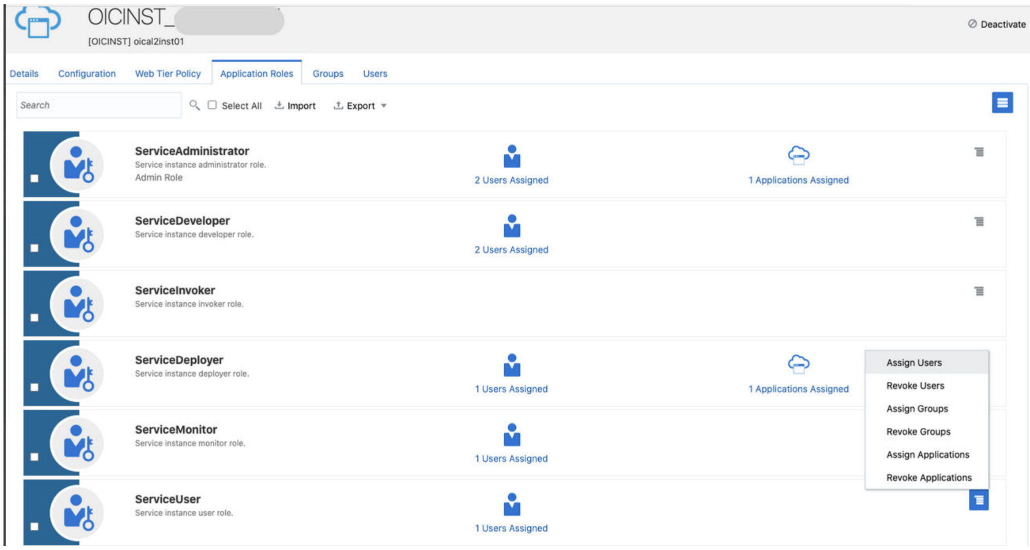
Secondary Audiences Add

Secondary Audience	Protected
urn:opc:lbaas:logicalguid=[redacted]	Yes

Scopes Add

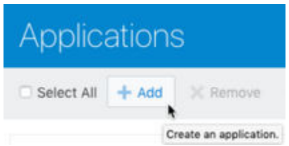
Scope	Protected	Description	Requires Consent
/ic/api/	Yes	OIC REST endpoint scope	false
urn:opc:resource:consumer::all	Yes	1 PaaS Scope	false

3. Add the appropriate users to the various Oracle Integration roles. For standard/ production configurations, use the **ServiceUser** role. (See Oracle Integration Service Roles in *Provisioning and Administering Oracle Integration 3*.)
4. To assign the user, go to the **Application Roles** section of the application.

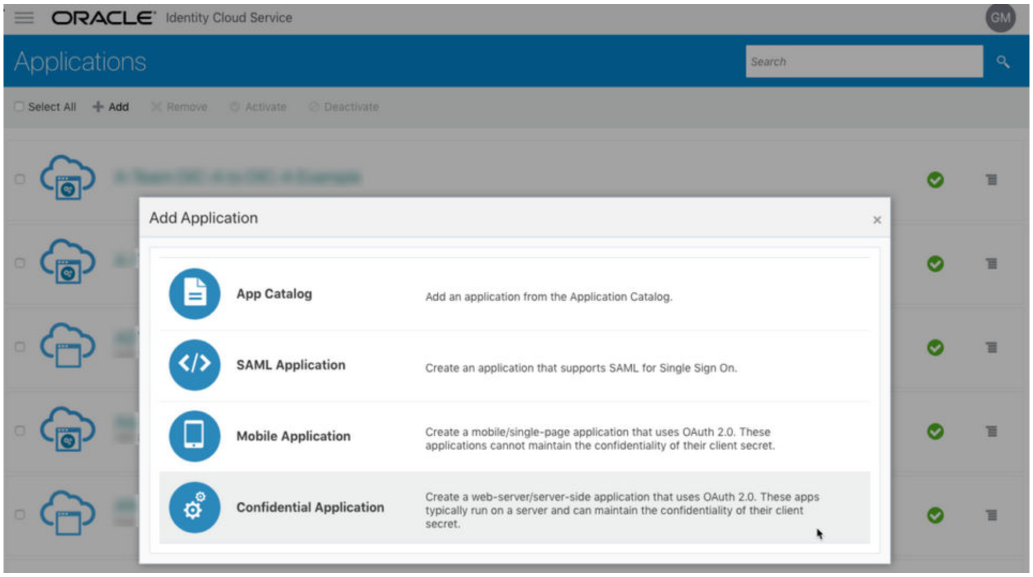


Role	Users	Applications
ServiceAdministrator Service instance administrator role. Admin Role	2 Users Assigned	1 Applications Assigned
ServiceDeveloper Service instance developer role.	2 Users Assigned	
ServiceInvoker Service instance invoker role.		
ServiceDeployer Service instance deployer role.	1 Users Assigned	1 Applications Assigned
ServiceMonitor Service instance monitor role.	1 Users Assigned	
ServiceUser Service instance user role.	1 Users Assigned	

- Configure the client application:
  To allow you to trigger the Oracle Integration integration with OAuth, the client application is required.
  1. In the Oracle Identity Cloud Service Console, go to the **Applications** section to create a new application that allows you to trigger the Oracle Integration integration with OAuth.



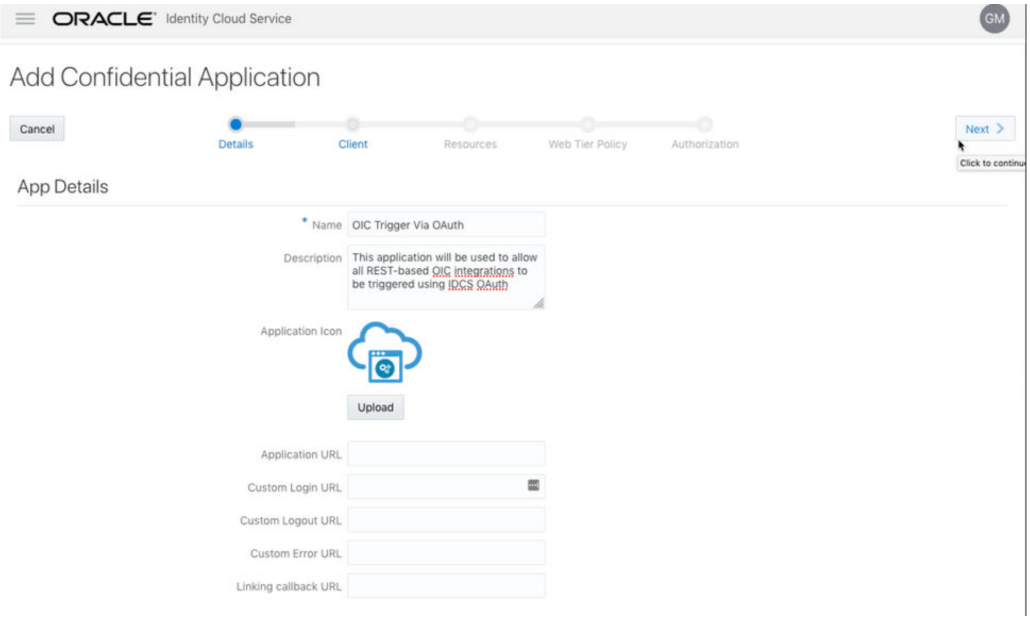
2. Select **Confidential Application**.



Add Application

App Catalog	Add an application from the Application Catalog.
SAML Application	Create an application that supports SAML for Single Sign On.
Mobile Application	Create a mobile/single-page application that uses OAuth 2.0. These applications cannot maintain the confidentiality of their client secret.
Confidential Application	Create a web-server/server-side application that uses OAuth 2.0. These apps typically run on a server and can maintain the confidentiality of their client secret.

3. Complete the **Details** page, and go to the **Client** page.



* Name: OIC Trigger Via OAuth

Description: This application will be used to allow all REST-based OIC integrations to be triggered using IDCS OAuth

Application Icon

Application URL

Custom Login URL

Custom Logout URL

Custom Error URL

Linking callback URL

4. On the **Client** page, select **Configure this application as a client now** and add the following.

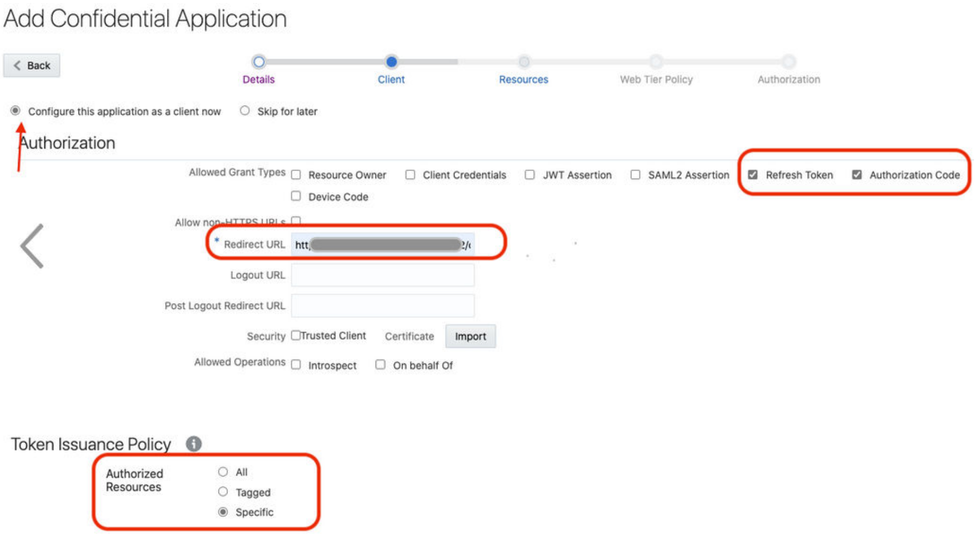
   a. Select **Refresh Token** and **Authorization Code** for **Allowed Grant Types**.

b. Set the redirect URL to the URL of the client application. After user login, Oracle Identity Cloud Service redirects to this URL with the authorization code.

> **Note:**
>
> If you don't know the following information, check with your administrator:
>
> – If your instance is new or upgraded from Oracle Integration Generation 2 to Oracle Integration 3.
>
> – The complete instance URL with the region included (required for new instances).

For Connections…	Include the Region as Part of the Redirect URL?	Example of Redirect URL to Specify…
Created on new Oracle Integration 3 instances	Yes.	`https://` `OIC_instance_URL.region.ocp.oraclecloud.com/icsapis/agent/oauth/callback`
Created on instances upgraded from Oracle Integration Generation 2 to Oracle Integration 3	No. This applies to both: – New connections created after the upgrade – Existing connections that were part of the upgrade	`https://` `OIC_instance_URL.ocp.oraclecloud.com/icsapis/agent/oauth/callback`

c. Select **Specific** in the **Authorized Resources** section.



Add Confidential Application

Back

Details — Client — Resources — Web Tier Policy — Authorization

● Configure this application as a client now ○ Skip for later

Authorization

Allowed Grant Types: □ Resource Owner □ Client Credentials □ JWT Assertion □ SAML2 Assertion ☑ Refresh Token ☑ Authorization Code □ Device Code

Allow non-HTTPS URLs □

* Redirect URL: htt[redacted]2/(

Logout URL

Post Logout Redirect URL

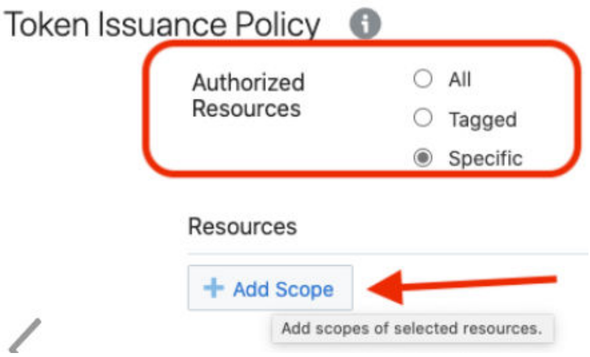
Security: □ Trusted Client Certificate Import

Allowed Operations: □ Introspect □ On behalf Of

Token Issuance Policy

Authorized Resources: ○ All ○ Tagged ● Specific

d. Click **Add Scope** under the **Resources** section.

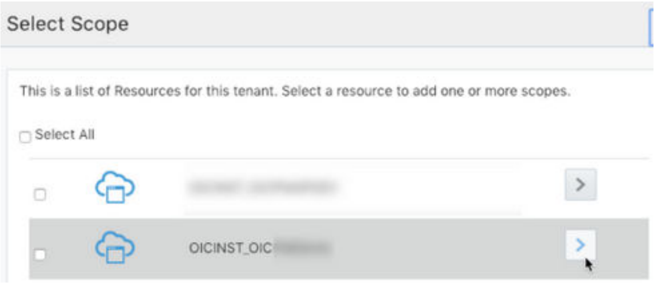


Token Issuance Policy

Authorized Resources: ○ All ○ Tagged ● Specific

Resources

e. Find the Oracle Integration application, and click **>**.



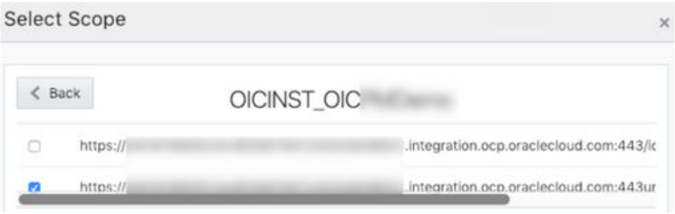
Select Scope

This is a list of Resources for this tenant. Select a resource to add one or more scopes.

□ Select All

□ [redacted]	>
□ OICINST_OIC[redacted]	>

f. Add the scope containing **urn:opc:resource:consumer::all**, and click **>**.



Select Scope ×

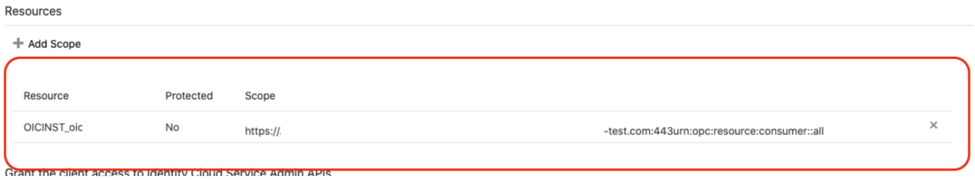
< Back

OICINST_OIC[redacted]

□ https://[redacted].integration.ocp.oraclecloud.com:443/ic

☑ https://[redacted].integration.ocp.oraclecloud.com:443ur

The scope containing **urn:opc:resource:consumer::all** is added.



Resources

Resource	Protected	Scope	
OICINST_oic	No	https://. -test.com:443urn:opc:resource:consumer::all	×

Grant the client access to Identity Cloud Service Admin APIs

g. Save your changes.

5. Click through the remaining wizard pages without making changes and save the application.

6. Activate the application for use.

- Validate the client application:

1. To fetch the authorization code, make the following request from the browser.

```
##Syntax
GET https://<IDCS-Service-Instance>.identity.oraclecloud.com/oauth2/v1/
```

```
authorize?client_id=<client-id>&response_type=code&redirect_uri=<client-
redirect-uri>&scope=<app_scope>%20offline_access&nonce=<nonce-
value>&state=<unique_value>

###where
#### <client-id> - ID of Client application generated.
#### <client-redirect-uri> - Redirect URI, in client application.
#### <app_scope> - scope added while creating application in client
configuration. (Ends with urn:opc:resource:consumer::all)
#### nonce - Optional, unique value to mitigate replay attacks
#### state - Recommended, Opaque to IDCS. Value used to maintain state
between the request and the callback
##Example
GET https://<idcs-host>/oauth2/v1/authorize?
client_id=<clientID>&response_type=code&redirect_uri=https://
app.getpostman.com/oauth2/callback&scope=https://
<Resource_APP_Audience>urn:opc:resource:consumer::all%20offline_access&n
once=121&state=12345544
```

2. If the user is not already logged in, Oracle Identity Cloud Service challenges the user to authenticate. Oracle Identity Cloud Service checks the user's credentials. (For authentication, the user assigned the **ServiceUser** role must be used.)
   After authentication is successful, Oracle Identity Cloud Service redirects back to the client redirect URL with the authorization code and state added to the URL.

   ```
   ##Response URL
   https://<redirect_URL>?code=<code_value>=&state=<state_value>

   ###Client should validate state received is same as one sent in request.
   ```

3. Capture the code value from the above response and make the following request to Oracle Identity Cloud Service to get the access token.

   ```
   ##Syntax
   curl -i -H 'Authorization: Basic <base64-clientid-secret>' -H 'Content-
   Type: application/x-www-form-urlencoded;charset=UTF-8' --request POST
   https://<IDCS-Service-Instance>.identity.oraclecloud.com/oauth2/v1/
   token -d 'grant_type=authorization_code&code=<authz-
   code>&redirect_uri=<client-redirect-uri>

   ###where
   #### <base64-clientid-secret> - BAse 64 encode clientId:ClientSecret
   #### <authz-code> - code value received as response on redirect.
   #### <client-redirect-uri> - Redirect URI, in client application.

   ##Example
   curl -i -H 'Authorization: Basic MDMx..NGY1' -H 'Content-Type:
   application/x-www-form-urlencoded;charset=UTF-8' --request POST https://
   <idcs_host>/oauth2/v1/token -d
   'grant_type=authorization_code&code=AQAg...3jKM4Gc=&redirect_uri=https:/
   /app.getpostman.com/oauth2/callback
   ```

4. Capture the `access_token` and `refresh_token` from the response.

   ```
   {
       "access_token": "eyJ4NXQjG...dfsdfsFgets2ed",
       "token_type": "Bearer",
       "expires_in": 3600,
       "refresh_token": "AQAgY2MzNjVlOTVhOTRh...vM5S0MkrFSpzc="
   }
   ```

5. Use the `access_token` in the authorization header to invoke the Oracle Integration trigger endpoint.

   ```
   curl --location --request GET 'https://OIC host/OIC endpoint' \
   --header 'Authorization: Bearer eyJ4NXQjG...dfsdfsFgets2ed'
   ```

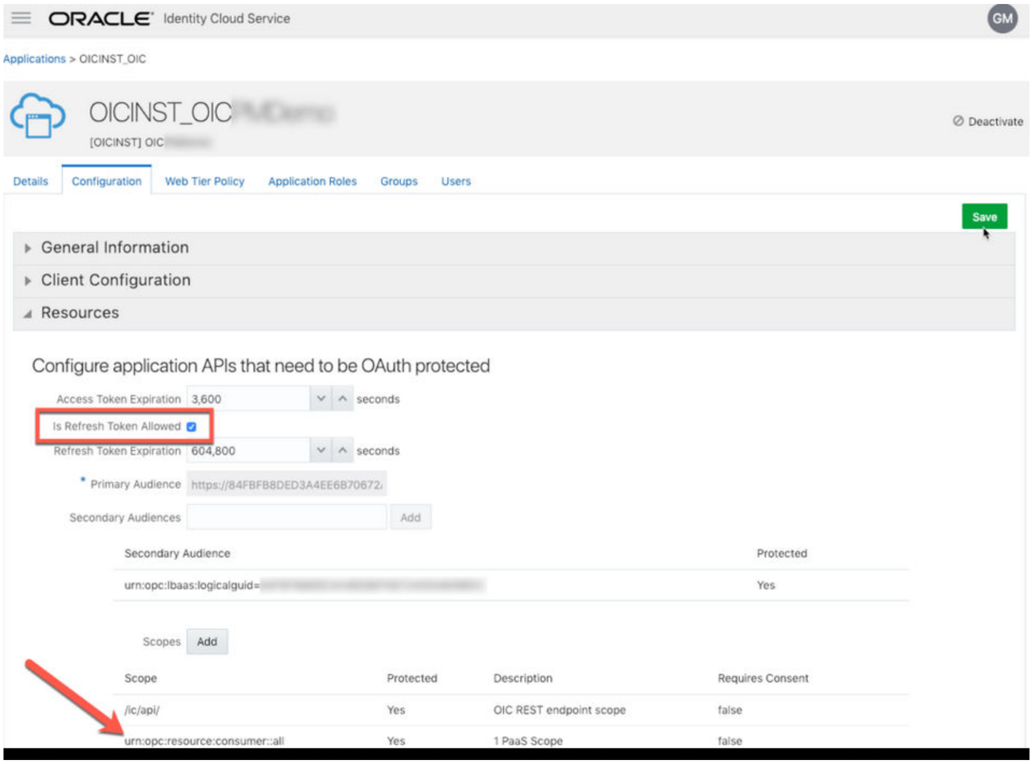
6. To update the access token, use the refresh token and make the request to Oracle Identity Cloud Service.
7. Capture the `access_token` and `refresh_token` from a response for further use.

   ```
   curl -i -H 'Authorization: Basic <base64-clientid-secret>' -H 'Content-
   Type: application/x-www-form-urlencoded;charset=UTF-8' --request POST
   https://<IDCS-Service-Instance>.identity.oraclecloud.com/oauth2/v1/
   token  -d 'grant_type=refresh_token&refresh_token=<refresh_token>'


   ##Example
   curl -i -H 'Authorization: Basic OGQyM...ZDA0Mjcz' -H 'Content-Type:
   application/x-www-form-urlencoded;charset=UTF-8' --request POST https://
   IDCS-Service-Instance.identity.oraclecloud.com/oauth2/v1/token  -d
   'grant_type=refresh_token&refresh_token=AQAgY2MzNjVlOTVhOTRh...vM5S0MkrF
   Spzc='
   ```

Prerequisites for Resource Owner Password Credentials

Perform the following tasks.

- Validate the Oracle Integration application and user roles:
  1. Verify that the **Is Refresh Token Allowed** option is enabled for the Oracle Integration Oracle Identity Cloud Service application.
  2. Check under the **Configuration** > **Resources** section of **Applications**. Note also that there is a special predefined scope (**urn:opc:resource:consumer::all**) that allows the triggering of integrations with OAuth.



OICINST_OIC[redacted]

[OICINST] OIC[redacted]

Configure application APIs that need to be OAuth protected

Access Token Expiration 3,600 seconds

Is Refresh Token Allowed ☑

Refresh Token Expiration 604,800 seconds

* Primary Audience https://84FBFB8DED3A4EE6B70672

Secondary Audiences Add

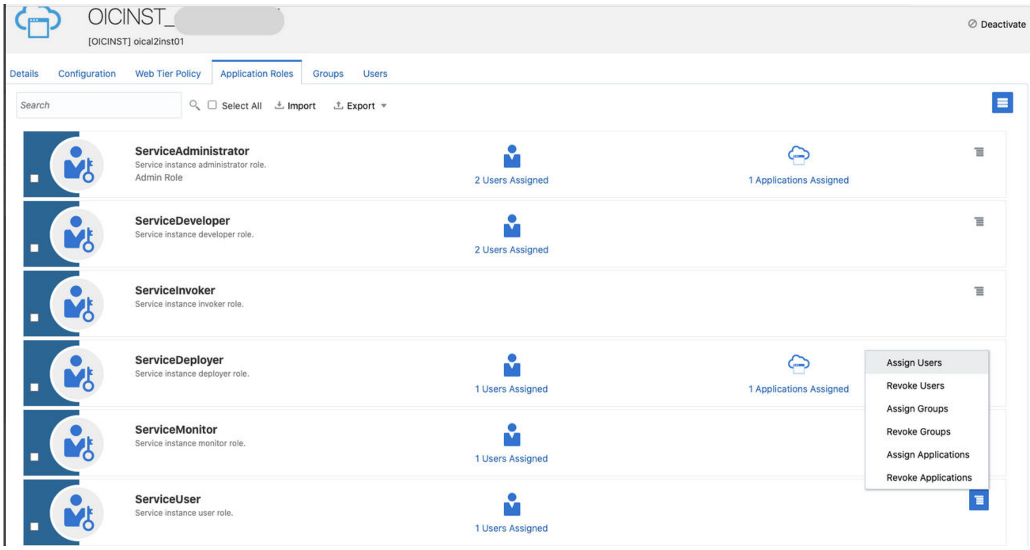
Secondary Audience	Protected
urn:opc:lbaas:logicalguid=[redacted]	Yes

Scopes Add

Scope	Protected	Description	Requires Consent
/ic/api/	Yes	OIC REST endpoint scope	false
urn:opc:resource:consumer::all	Yes	1 PaaS Scope	false

3. Add the appropriate users to the various Oracle Integration roles. For standard/ production configurations, use the **ServiceUser** role. (See Oracle Integration Service Roles in *Provisioning and Administering Oracle Integration 3*.)

4. To assign the user, go to the **Application Roles** section of the application.



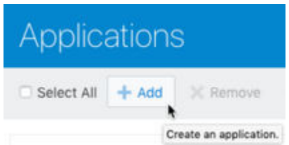
OICINST_[redacted]

[OICINST] oical2inst01

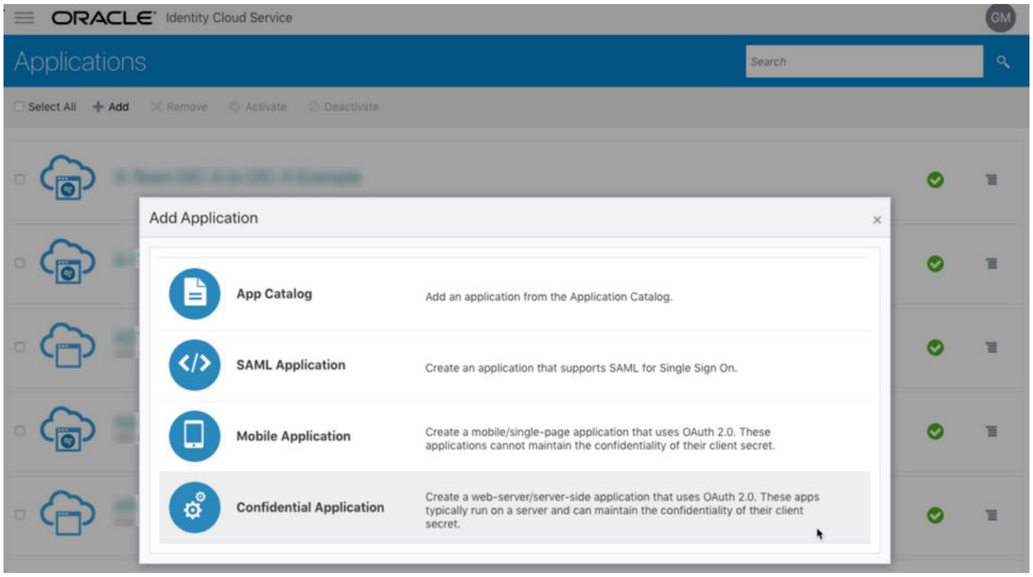
Role	Users	Applications
ServiceAdministrator Service instance administrator role. Admin Role	2 Users Assigned	1 Applications Assigned
ServiceDeveloper Service instance developer role.	2 Users Assigned	
ServiceInvoker Service instance invoker role.		
ServiceDeployer Service instance deployer role.	1 Users Assigned	1 Applications Assigned
ServiceMonitor Service instance monitor role.	1 Users Assigned	
ServiceUser Service instance user role.	1 Users Assigned	

- Configure the client application:
  To trigger the integration with OAuth, a client application is required.

  1. In the Oracle Identity Cloud Service Console, go to the **Applications** section to create a new application that allows you to trigger an integration with OAuth.



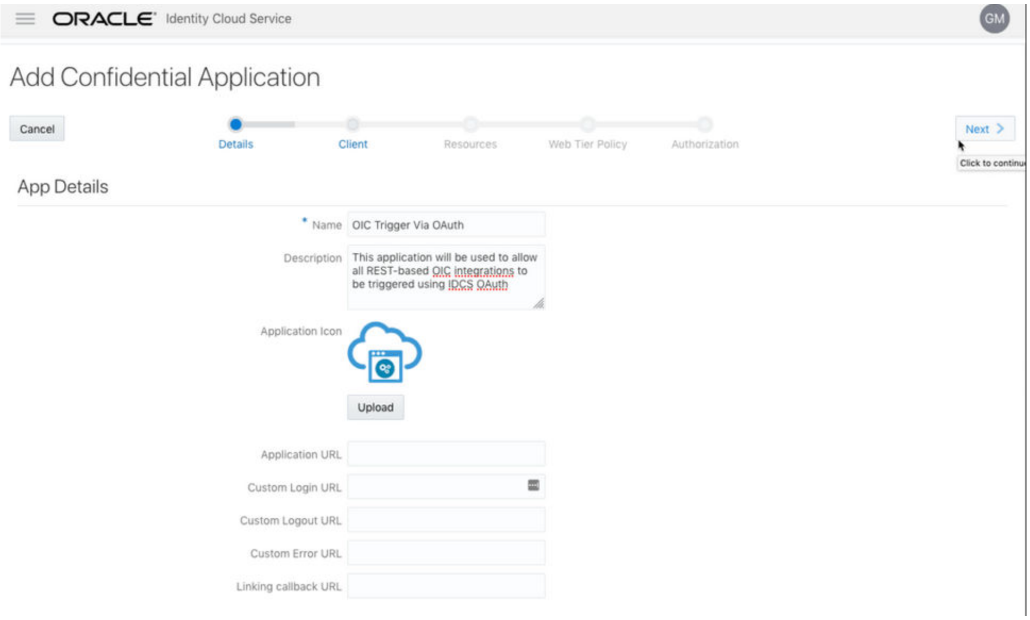
2. Click **Add**.
3. Select **Confidential Application**.



Add Application

App Catalog	Add an application from the Application Catalog.
SAML Application	Create an application that supports SAML for Single Sign On.
Mobile Application	Create a mobile/single-page application that uses OAuth 2.0. These applications cannot maintain the confidentiality of their client secret.
Confidential Application	Create a web-server/server-side application that uses OAuth 2.0. These apps typically run on a server and can maintain the confidentiality of their client secret.

4. Complete the **Details** page, and go to the **Client** page.



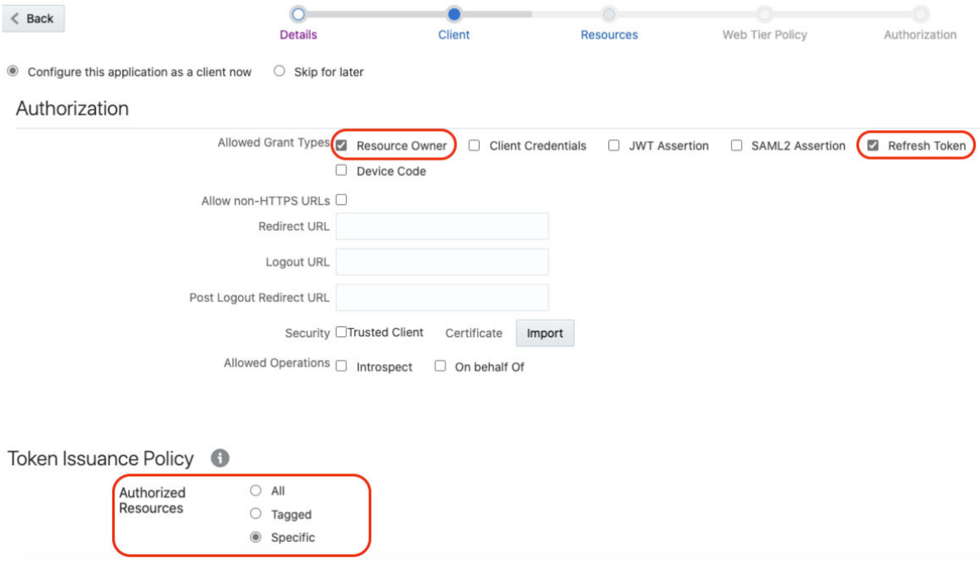
Add Confidential Application

App Details

* Name	OIC Trigger Via OAuth
Description	This application will be used to allow all REST-based OIC integrations to be triggered using IDCS OAuth
Application Icon	
Application URL	
Custom Login URL	
Custom Logout URL	
Custom Error URL	
Linking callback URL	

5. On the **Client** page, select **Configure this application as a client now** and add the following.
   a. Select **Resource Owner** and **Refresh Token** for **Allowed Grant Types**.

b. Select **Specific** in the **Authorized Resources** section.



● Configure this application as a client now ○ Skip for later

Authorization

Allowed Grant Types ☑ Resource Owner □ Client Credentials □ JWT Assertion □ SAML2 Assertion ☑ Refresh Token □ Device Code

Allow non-HTTPS URLs □

Redirect URL

Logout URL

Post Logout Redirect URL

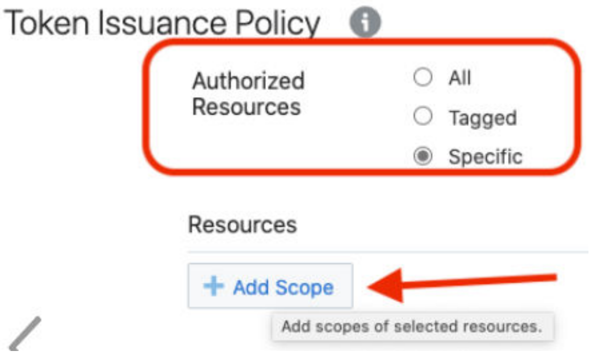
Security □ Trusted Client Certificate Import

Allowed Operations □ Introspect □ On behalf Of

Token Issuance Policy

Authorized Resources ○ All ○ Tagged ● Specific

c. Click **Add Scope** under the **Resources** section.

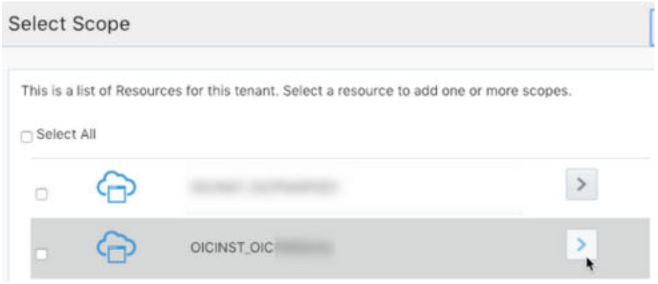


Token Issuance Policy

Authorized Resources ○ All ○ Tagged ● Specific

Resources

d. Find the Oracle Integration application.



Select Scope

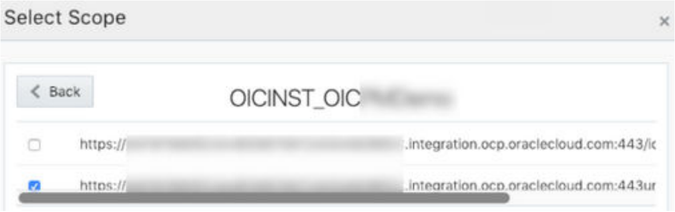
This is a list of Resources for this tenant. Select a resource to add one or more scopes.

□ Select All

□ [redacted]

□ OICINST_OIC[redacted]

e. Add the scope containing **urn:opc:resource:consumer::all**, and click **>**.



Select Scope

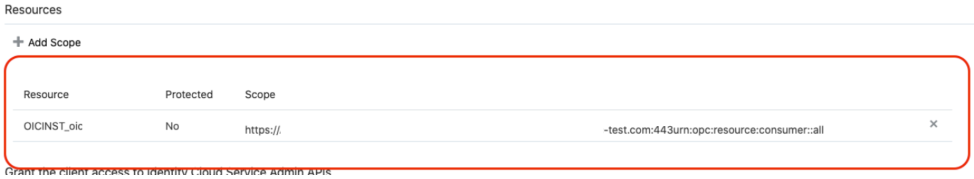
< Back

OICINST_OIC[redacted]

□ https://[redacted].integration.ocp.oraclecloud.com:443/ic

☑ https://[redacted].integration.ocp.oraclecloud.com:443ur

The scope containing **urn:opc:resource:consumer::all** is added.



Resources

+ Add Scope

Resource	Protected	Scope
OICINST_oic	No	https://. -test.com:443urn:opc:resource:consumer::all

Grant the client access to Identity Cloud Service Admin APIs

f. Save your changes.

6. Click through the remaining wizard pages without making changes and save the application.
7. Activate the application for use.

- Validate the client application:

1. To fetch the access client, make a request to Oracle Identity Cloud Service with the user name and password in the payload.

```
##Syntax
curl -i -H 'Authorization: Basic <base64Encoded_clientid:secret>' -H
'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' --
request POST https://<IDCS-Service-Instance>.identity.oraclecloud.com/
oauth2/v1/token -d 'grant_type=password&username=<user-
name>&password=<password>&scope=<App_Scope>%20offline_access'

###where
#### <base64-clientid-secret> - Base 64 encode clientId:ClientSecret
#### <username> - user for token needs to be issued (must be in
serviceuser role).
#### <password> - password for above user
#### <app_scope> - Scope added while creating application in client
configuration section (Ends with urn:opc:resource:consumer::all)
##Example
curl -i -H 'Authorization: Basic OGQyM...ZDA0Mjcz' -H 'Content-Type:
application/x-www-form-urlencoded;charset=UTF-8' --request POST https://
<idcs_host>/oauth2/v1/token -d
'grant_type=password&username=sampleUser&password=SamplePassword&scope=h
ttps://
<Resource_APP_Audience>urn:opc:resource:consumer::all%20offline_access'
```

2. Capture the `access_token` and `refresh_token` from the response.

```
{
    "access_token": "eyJ4NXQjG...dfsdfsFgets2ed",
    "token_type": "Bearer",
    "expires_in": 3600,
    "refresh_token": "AQAgY2MzNjVlOTVhOTRh...vM5S0MkrFSpzc="
}
```

3. Use the `access_token` in the authorization header to invoke the Oracle Integration trigger endpoint.

   ```
   curl --location --request GET 'https://OIC host/OIC endpoint' \
   --header 'Authorization: Bearer eyJ4NXQjG...dfsdfsFgets2ed'
   ```

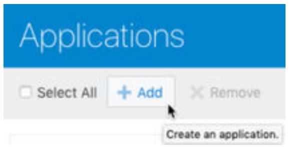
4. To update the access token, use the refresh token and make a request to Oracle Identity Cloud Service.
5. Capture the `access_token` and `refresh_token` from the response for further use.

   ```
   curl -i -H 'Authorization: Basic <base64-clientid-secret>' -H 'Content-
   Type: application/x-www-form-urlencoded;charset=UTF-8' --request POST
   https://<IDCS-Service-Instance>.identity.oraclecloud.com/oauth2/v1/
   token  -d 'grant_type=refresh_token&refresh_token=<refresh_token>'

   ##Example
   curl -i -H 'Authorization: Basic OGQyM...ZDA0Mjcz' -H 'Content-Type:
   application/x-www-form-urlencoded;charset=UTF-8' --request POST https://
   <IDCS-Service-Instance>.identity.oraclecloud.com/oauth2/v1/token  -d
   'grant_type=refresh_token&refresh_token=AQAgY2MzNjVlOTVhOTRh...vM5S0MkrF
   Spzc='
   ```

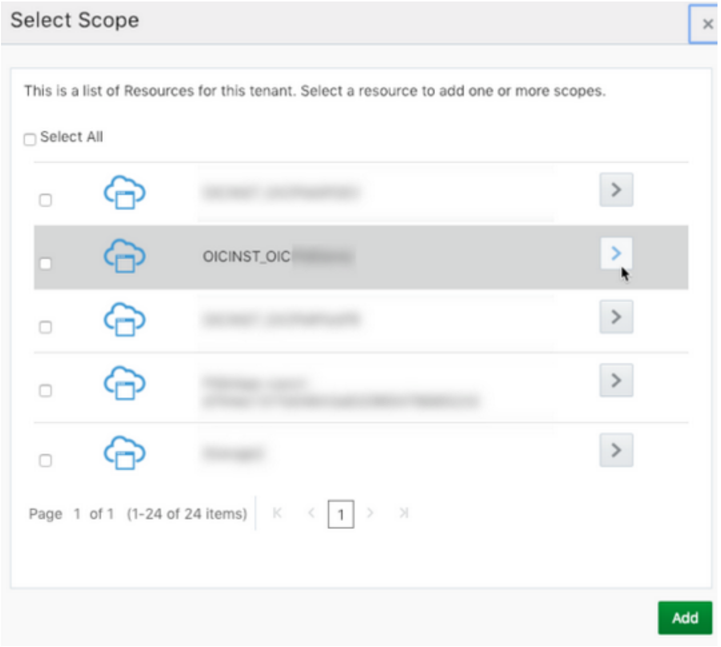
Prerequisites for Client Credentials

- Configure the client application.

1. In the Oracle Identity Cloud Service Console, go to the **Applications** section to create a new application that allows you to trigger an integration with OAuth.



2. Click **Add**.
3. Select **Confidential Application**.
4. Complete the **Details** page, and click **Next**.
5. On the **Client** page, select **Configure this application as a client now**, and complete the following:
   a. Select **Client Credentials** from the **Allowed Grant Types** list.
   b. Select **Specific** in the **Authorized Resources** area of the **Token Issuance Policy** section.
   c. Click **Add Scope** under the **Resources** section.
   d. Find the Oracle Integration application, and click **>**.



Select Scope ×

This is a list of Resources for this tenant. Select a resource to add one or more scopes.

□ Select All

□	[redacted]	>
□	OICINST_OIC[redacted]	>
□	[redacted]	>
□	[redacted]	>
□	[redacted]	>

Page 1 of 1 (1-24 of 24 items) 1

Add

e. Add the scope containing **urn:opc:resource:consumer::all**.

f. Save your changes.

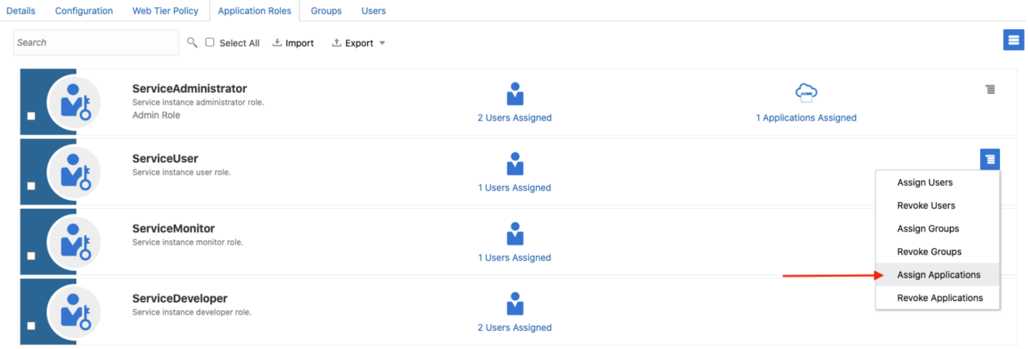
6. Click through the remaining wizard pages without making changes and save the application.

7. Activate the application for use.

- Add roles to the client application.

1. Go to the **Application Roles** tab of the Oracle Identity Cloud Service application.

2. Select **Assign Applications** for the **ServiceUser** role.



Details | Configuration | Web Tier Policy | Application Roles | Groups | Users

Search | □ Select All | Import | Export

Role	Users	Applications
ServiceAdministrator Service instance administrator role. Admin Role	2 Users Assigned	1 Applications Assigned
ServiceUser Service instance user role.	1 Users Assigned	
ServiceMonitor Service instance monitor role.	1 Users Assigned	
ServiceDeveloper Service instance developer role.	2 Users Assigned	

- Validate the client application.

1. Fetch the access client to make an access token request to Oracle Identity Cloud Service with the client credentials.

```
##Syntax
curl -i -H 'Authorization: Basic <base64Encoded clientid:secret>' -H
'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' --
request POST https://<IDCS-Service-Instance>.identity.oraclecloud.com/
oauth2/v1/token -d 'grant_type=client_credentials&scope=<app scope>'
```

```
###where
#### <base64-clientid-secret> - Base 64 encode clientId:ClientSecret
#### <app scope> - Scope added while creating application in client
configuration section (Ends with urn:opc:resource:consumer::all)

##Example
curl -i -H 'Authorization: Basic OGQyM...ZDA0Mjcz' -H 'Content-Type:
application/x-www-form-urlencoded;charset=UTF-8' --request POST https://
<idcs_host>/oauth2/v1/token -d
'grant_type=client_credentials&scope=https://<Resource APP
Audience>urn:opc:resource:consumer::all'
```

2. Capture the `access_token` from the response.

   ```
   {
       "access_token": "eyJ4NXQjG...dfsdfsFgets2ed",
       "token_type": "Bearer",
       "expires_in": 3600
   }
   ```

3. Use the `access_token` in the authorization header to invoke the trigger endpoint.

   ```
   curl --location --request GET 'https://OIC host/OIC endpoint' \
   --header 'Authorization: Bearer eyJ4NXQjG...dfsdfsFgets2ed'
   ```

Authentication Types

The REST Adapter supports the following types of authentication:

- For scenarios when the REST Adapter invokes an external REST endpoint:
  - Basic Authentication
  - OAuth Client Credentials (two-legged flow)
  - OAuth Resource Owner Password Credentials (two-legged flow)
  - OAuth Authorization Code Credentials (three-legged flow)
  - OAuth Custom Three Legged Flow
  - OAuth Custom Two Legged Flow
  - API Key Based Authentication
  - OAuth 1.0 One Legged Authentication
  - Amazon Web Services (AWS) Signature Version 4
  - Oracle Cloud Infrastructure (OCI) Signature Version 1
  - OAuth Client Credentials using JWT Client Assertion
  - OAuth using JWT User Assertion
  - OCI Service Invocation
- For scenarios when the REST Adapter is used to create a REST endpoint to trigger an integration:
  - Basic Authentication
  - OAuth 2.0

  - OAuth 2.0 or Basic Authentication

See Configure Connection Security for more information about these security policies.

Role-Based Connections

The REST Adapter is bidirectional. You can configure the REST Adapter depending on the context in which you want to use the connection.

- Trigger: The REST Adapter is used to create a REST endpoint to trigger an integration. You select **Trigger** from the **Role** list on the Create New Connection dialog. When configured as a trigger, a base URI is not required. The security policy defined in the inbound direction accepts credentials configured in the identity domain. Therefore, you are not required to provide the applicable credentials. When configuring security on the Connections page, you only provide the security policy that must be attached to the inbound endpoint. The following security policies are available:
  - Basic authentication
  - OAuth 2.0
  - Basic authentication and OAuth 2.0

  Agent configuration is not applicable on a connection with the trigger role.
- Invoke: The REST Adapter is used to invoke external REST endpoints. A base URI and security configuration for accessing external protected resources are required. You are prompted for these additional details on the Connections page. You cannot use an invoke connection on the trigger side.
- Trigger and invoke: The REST Adapter is used in both the trigger and invoke directions of an integration. This connection requires invoke and trigger values. Basic Authentication can be used in both trigger and invoke connections.

See Create a Connection.

Extensibility Support for Multiple OAuth Providers

You can use the extensibility framework of the REST Adapter to access the OAuth-protected resource of endpoints. This framework enables you to access endpoints that have implemented their own variations of OAuth.

The OAuth standard provides flexibility for endpoints to define specific aspects of their OAuth flows. For example:

- Create their own properties.
- Decide when to use these properties in an OAuth flow. For example, some custom properties may be required with the authorization request, while others may be required for the access token request or for the refresh of the access token after its expiration.
- Decide how to pass these properties in an OAuth flow. For example, whether a property is passed as a header, query parameter, or payload.

To address these challenges, Oracle Integration provides two custom security policies that enable you to specify each step in the OAuth flow when you create the REST Adapter connection:

- OAuth custom two-legged flow: The client application directly interacts with the authorization server on behalf of a resource owner.

- OAuth custom three-legged flow: The client application redirects the owner to a separate resource URL where the resource owner authenticates and provides consent for the flow to continue.

This enables you to adapt to most OAuth framework scenarios and integrate with many third-party applications without writing additional code.

- During design-time, the access token is obtained, validated, and stored in the CSF. The security token is also stored in the CSF.
- During runtime, the access token is retrieved, applied, and managed. A valid access token is applied to the request before invoking the REST endpoint.

Specify the OAuth custom two-legged flow and three-legged flow security policies. See Configure Connection Security and REST Adapter Use Cases.

> **Note:**
>
> This extensibility feature is an advanced feature, and not for business users. Users of this feature should use a tool such as postman to configure the necessary properties.

REST API Support

The following sections describe REST Adapter REST API capabilities in more detail.

Topics:

- Oracle Cloud Infrastructure REST API Support with the OCI Signature Version 1 Security Policy
- On-Premises REST API Support with the Agent

Oracle Cloud Infrastructure REST API Support with the OCI Signature Version 1 Security Policy

You can call the Oracle Cloud Infrastructure (OCI) REST API by configuring the REST Adapter connection to use the OCI Signature Version 1 security policy.

OCI signature support in the REST Adapter enables a user to use Oracle Cloud Infrastructure services. For example, you can create an integration that calls Oracle Cloud Infrastructure to create a storage bucket.

See Configure Connection Security.

On-Premises REST API Support with the Agent

The REST Adapter-specific connection can be configured to use the connectivity agent to consume REST APIs that are hosted either on a customer's on-premises network or on a private cloud, are truly private APIs, and are not exposed as public APIs that can be consumed over the internet.

Oracle Integration provides the agent framework to enable you to create integrations and exchange messages between on-premises applications and Oracle Integration. You can integrate on-premises REST APIs with Oracle Integration with the on-premises connectivity agent. Once you create an agent group and install the on-premises agent, you can create and configure a REST Adapter connection on the Connections page as follows:

- Select **REST API Base URL** from the **Connection Type** list and enter the appropriate URL in the **Connection URL** field. No other connection types are supported.
- Select **Basic Authentication**, **OAuth Client Credentials**, **OAuth Resource Owner Password Credentials**, **OAuth Custom Two Legged Flow**, or **No Security Policy** from the **Security Policy** list. No other security policies are supported.
- Select the previously-created agent group in the Select an Agent Group dialog.
- Note that the token server cannot be on-premises. If the provider is accessible through the cloud, you must get the bearer token in the cloud and perform the agent-based request invocation by setting the authorization header.

For conceptual information about the on-premises agent, see About Creating Hybrid Integrations Using Oracle Integration. For information about creating an agent group and installing the on-premises agent, see Manage the Agent Group and the On-Premises Connectivity Agent.

OpenAPI Support

The following sections describe REST Adapter OpenAPI capabilities in more detail.

Topics:

- Support of Polymorphic Constructs for OpenAPI Connectivity
- Support for OpenAPI Documents with External References
- Support for Publishing Interfaces for Oracle Integration Flows as OpenAPI Documents
- Consumption of OpenAPI Multipart for JSON and Form Data

Support of Polymorphic Constructs for OpenAPI Connectivity

OpenAPI 3.0 provides the `allOf`, `oneOf`, and `anyOf` keywords to use for combining schemas. You can use these keywords to create a complex schema or validate a value against multiple criteria. If an OpenAPI document has schemas that use the `allOf`, `oneOf`, or `anyOf` keyword or a combination of any, Oracle Integration can process those schemas. The following sections describe the patterns supported by `allOf`, `oneOf`, and `anyOf` in Oracle Integration. Any pattern not listed should be considered unsupported.

Keyword	Description	See
`allOf`	Validates the value against *all* of the subschemas.	allOf Keyword Pattern Support
`oneOf`	Validates the value against *exactly one* of the subschemas.	oneOf Keyword Pattern Support
`anyOf`	Validates the value against *any* of the subschemas.	anyOf Keyword Pattern Support

allOf Keyword Pattern Support

Use the `allOf` keyword to combine and extend model definitions. The `allOf` keyword takes an array of object definitions used for independent validation, but together compose a single object.

Note:

The allOf pattern with a subschema being of a simple type is not currently supported.
Any other patterns not listed in the following sections should also be considered unsupported.

Oracle Integration supports the following allOf keyword patterns.

- allOf with All Subschemas Defined as $ref
- allOf with All Subschemas Defined Inline
- allOf with a Mix of Inline and $ref Subschemas
- allOf Defined as Item of an Array Type
- allOf Defined as Items of a Top Level Array
- allOf with a Nested allOf Subschema Defined as $ref
- allOf with a Nested allOf Subschema Defined Inline
- allOf with a Nested oneOf Subschema Defined as $ref
- allOf with a Nested oneOf Subschema Defined Inline
- allOf with a Nested anyOf Subschema Defined as $ref
- allOf with a Nested anyOf Subschema Defined Inline

allOf with All Subschemas Defined as $ref

You can use an allOf pattern in which all subschemas are defined as references ($ref). For this example, there are three subschemas defined as references.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "$ref": "#/components/schemas/ItemProperties"
                    },
                    {
                        "$ref": "#/components/schemas/ItemProductionPrivateVO-item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemProductionPrivateVO-item-response-forChildren"
                    }
                ]
            }
        }
    }
}
```

allOf with All Subschemas Defined Inline

You can use an `allOf` pattern in which all subschemas are defined inline. For this example, there are two subschemas defined inline.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "required": [
                            "count",
                            "hasMore"
                        ],
                        "type": "object",
                        "properties": {
                            "totalResults": {
                                "type": "integer",
                                "format": "int32"
                            },
                            "count": {
                                "type": "integer",
                                "format": "int32"
                            },
                            "hasMore": {
                                "type": "boolean"
                            }
                        }
                    },
                    {
                        "type": "object",
                        "properties": {
                            "items": {
                                "title": "Items",
                                "type": "array",
                                "description": "The items in the collection.",
                                "items": {
                                    "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-response"
                                },
                                "x-cardinality": "1"
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

allOf with a Mix of Inline and $ref Subschemas

You can use an `allOf` pattern in which the subschemas are defined both as references (`$ref`) and inline. For this example, there is one subschema defined as a reference and one subschema defined inline.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "$ref": "#/components/schemas/CollectionProperties"
                    },
                    {
                        "type": "object",
                        "properties": {
                            "items": {
                                "title": "Items",
                                "type": "array",
                                "description": "The items in the collection.",
                                "items": {
                                    "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-response"
                                },
                                "x-cardinality": "1"
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

allOf Defined as Item of an Array Type

You can use an `allOf` pattern defined as an item of an array type.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "properties": {
                    "repeatingElement": {
                        "description": "The items in the collection.",
                        "items": {
                            "allOf": [
                                {
                                    "$ref": "#/components/schemas/
ItemProperties"
                                },
                                {
                                    "$ref": "#/components/schemas/
ItemProductionPrivateVO-item"
```

```
                                }
                            ]
                        },
                        "title": "repeatingElement",
                        "type": "array",
                        "x-cardinality": "1"
                    },
                    "someStringElement": {
                        "type": "string"
                    }
                },
                "type": "object"
            }
        }
    }
}
```

allOf Defined as Items of a Top Level Array

You can use an `allOf` pattern defined as items of a top level array.

```
{
    "components": {
        "schemas": {
            "Pets": {
                "description": "The items in the collection.",
                "items": {
                    "allOf": [
                        {
                            "$ref": "#/components/schemas/Cat_Type"
                        },
                        {
                            "$ref": "#/components/schemas/Dog_Type"
                        }
                    ]
                },
                "title": "Pets",
                "type": "array",
                "x-cardinality": "1"
            }
        }
    }
}
```

allOf with a Nested allOf Subschema Defined as $ref

You can use an `allOf` pattern in which another `allOf` is nested and defined as a reference (`$ref`).

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
patch-item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
item-patch-request-forChildren"
                    }
                ]
            },
            "ItemRootIccPrivateVO-patch-item": {
                "allOf": [
                    {
                        "$ref": "#/components/schemas/ItemEffCategoryVO-patch-
item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
updatableFields"
                    }
                ],
                "title": "Item Extensible Flexfield"
            },
            "ItemRootIccPrivateVO-item-patch-request-forChildren": {
                "type": "object",
                "properties": {
                    "ItemEFFBItem__Details__EFFPrivateVO": {
                        "type": "array",
                        "items": {
                            "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-patch-request"
                        },
                        "x-cardinality": "1"
                    }
                }
            }
        }
    }
}
```

allOf with a Nested allOf Subschema Defined Inline

You can use an `allOf` pattern in which another `allOf` is nested and defined as a reference (`$ref`).

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "allOf": [
                            {
                                "$ref": "#/components/schemas/
ItemEffCategoryVO-patch-item"
                            },
                            {
                                "$ref": "#/components/schemas/
ItemRootIccPrivateVO-updatableFields"
                            }
                        ],
                        "title": "Item Extensible Flexfield"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
item-patch-request-forChildren"
                    }
                ]
            }
        }
    }
}
```

allOf with a Nested oneOf Subschema Defined as $ref

You can use an `allOf` pattern in which a referenced subschema is a nested `oneOff`.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "$ref": "#/components/schemas/ItemEffCategoryVO-patch-
item"
                    },
                    {
                        "type": "object",
                        "properties": {
                            "items": {
                                "title": "Items",
                                "type": "array",
                                "description": "The items in the collection.",
                                "items": {
                                    "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-response"
                                },
                                "x-cardinality": "1"
                            }
                        }
                    }
                ]
            },
            "ItemEffCategoryVO-patch-item": {
                "discriminator": {
                    "propertyName": "CategoryCode",
                    "mapping": {
                        "Production": "schemas/ItemProductionPrivateVO-item-
response",
                        "ROOT_ICC": "schemas/ItemRootIccPrivateVO-item-
response"
                    }
                },
                "oneOf": [
                    {
                        "$ref": "schemas/ItemProductionPrivateVO-item-
response"
                    },
                    {
                        "$ref": "schemas/ItemRootIccPrivateVO-item-response"
                    }
                ]
            }
        }
    }
}
```

allOf with a Nested oneOf Subschema Defined Inline

You can use an `allOf` pattern in which a `oneOf` is nested and defined inline.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "$ref": "#/components/schemas/ItemProperties"
                    },
                    {
                        "discriminator": {
                            "propertyName": "CategoryCode",
                            "mapping": {
                                "Production": "#/components/schemas/
ItemProductionPrivateVO-item-response",
                                "ROOT_ICC": "#/components/schemas/
ItemRootIccPrivateVO-item-response"
                            }
                        },
                        "oneOf": [
                            {
                                "$ref": "#/components/schemas/
ItemProductionPrivateVO-item-response"
                            },
                            {
                                "$ref": "#/components/schemas/
ItemRootIccPrivateVO-item-response"
                            }
                        ]
                    }
                ]
            }
        }
    }
}
```

allOf with a Nested anyOf Subschema Defined as $ref

You can use an `allOf` pattern in which a nested `anyOf` subschema is defined as a reference (`$ref`).

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
patch-item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
item-patch-request-forChildren"
                    }
                ]
            },
            "ItemRootIccPrivateVO-patch-item": {
                "anyOf": [
                    {
                        "$ref": "#/components/schemas/ItemEffCategoryVO-patch-
item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
updatableFields"
                    }
                ],
                "title": "Item Extensible Flexfield"
            },
            "ItemRootIccPrivateVO-item-patch-request-forChildren": {
                "type": "object",
                "properties": {
                    "ItemEFFBItem__Details__EFFPrivateVO": {
                        "type": "array",
                        "items": {
                            "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-patch-request"
                        },
                        "x-cardinality": "1"
                    }
                }
            }
        }
    }
}
```

allOf with a Nested anyOf Subschema Defined Inline

You can use an `allof` pattern in which a nested `anyOf` subschema is defined inline.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "allOf": [
                    {
                        "anyOf": [
                            {
                                "$ref": "#/components/schemas/
ItemEffCategoryVO-patch-item"
                            },
                            {
                                "$ref": "#/components/schemas/
ItemRootIccPrivateVO-updatableFields"
                            }
                        ],
                        "title": "Item Extensible Flexfield"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
item-patch-request-forChildren"
                    }
                ]
            }
        }
    }
}
```

oneOf Keyword Pattern Support

Use the `oneOf` keyword to ensure that the given data is valid against one of the subschemas.

Note:

When using `oneOf`, the following guidelines are mandatory:

- The `discriminator` element must be present.
- The `discriminator` element name must be unique across all subschemas.

Note:

The following oneOf keyword patterns are not currently supported.

- oneOf with a subschema being of simple type
- oneOf with a nested oneOf subschema (either defined inline or as $ref)

Any other patterns not listed in the following sections should also be considered unsupported.

Oracle Integration supports the following oneOf keyword patterns.

- oneOf with All Subschemas Defined as $ref
- oneOf with All Subschemas Defined as Inline
- oneOf with a Mix of Inline and $ref Subschemas
- oneOf with a Subschema Containing Multiple Mappings
- oneOf Defined as Items of an Array Type
- oneOf Defined as Items of a Top Level Array
- oneOf with a Nested allOf Subschema Defined as $ref
- oneOf with a Nested allOf Subschema Defined Inline
- oneOf with a Nested anyOf Subschema Defined as $ref
- oneOf with a Nested anyOf Subschema Defined Inline

oneOf with All Subschemas Defined as $ref

You can use a oneOf pattern in which all subschemas are defined as references ($ref). For this example, the pet schema can validate against either of two subschemas.

```
{
    "schemas": {
        "Pet": {
            "discriminator": {
                "propertyName": "pet_type",
                "mapping": {
                    "CAT": "#/components/schemas/Cat_Type",
                    "DOG": "#/components/schemas/Dog_Type"
                }
            },
            "oneOf": [
                {
                    "$ref": "#/components/schemas/Cat_Type"
                },
                {
                    "$ref": "#/components/schemas/Dog_Type"
                }
            ]
        },
        "Cat_Type": {
            "type": "object",
            "required": [
                "pet_type"
            ],
            "properties": {
                "hunts": {
                    "type": "boolean"
                },
                "pet_type": {
                    "type": "string"
                },
                "age": {
                    "type": "integer"
                }
            }
        },
        "Dog_Type": {
            "type": "object",
            "required": [
                "pet_type"
            ],
            "properties": {
                "bark": {
                    "type": "boolean"
                },
                "pet_type": {
                    "type": "string"
                },
                "breed": {
                    "type": "string",
                    "enum": [
                        "Dingo",
                        "Husky",
                        "Retriever",
                        "Shepherd"
                    ]
                }
            }
        }
    }
}
```

oneOf with All Subschemas Defined as Inline

You can use a `oneOf` pattern in which all subschemas are defined inline. For this example, the `Pet` schema can validate against either of two subschemas.

> **Note:**
>
> This pattern is only supported for outgoing messages from Oracle Integration (that is, either an invoke connection request or a trigger connection response back to a client).

```
{
    "schemas": {
        "Pet": {
            "discriminator": {
                "propertyName": "pet_type"
            },
            "oneOf": [
                {
                    "properties": {
                        "age": {
                            "type": "integer"
                        },
                        "hunts": {
                            "type": "boolean"
                        },
                        "pet_type": {
                            "type": "string"
                        }
                    },
                    "required": [
                        "pet_type"
                    ],
                    "type": "object"
                },
                {
                    "properties": {
                        "bark": {
                            "type": "boolean"
                        },
                        "breed": {
                            "enum": [
                                "Dingo",
                                "Husky",
                                "Retriever",
                                "Shepherd"
                            ],
                            "type": "string"
                        },
                        "pet_type": {
                            "type": "string"
                        }
                    },
                    "required": [
                        "pet_type"
                    ],
                    "type": "object"
                }
            ]
        }
    }
}
```

oneOf with a Mix of Inline and $ref Subschemas

> **Note:**
>
> - `discriminator` mapping is mandatory in this pattern for all subschemas defined as `$ref`.
> - If more than one subschema is defined as inline, this pattern is only supported for outgoing messages from Oracle Integration (that is, either an invoke connection request or a trigger response back to the client).

You can use a `oneOf` pattern in which the subschemas are defined both as references (`$ref`) and inline. For this example, there is one subschema defined as a reference and one subschema defined inline.

```
{
    "schemas": {
        "Pet": {
            "discriminator": {
                "propertyName": "pet_type",
                "mapping": {
                    "CAT": "#/components/schemas/Cat_Type"
                }
            },
            "oneOf": [
                {
                    "$ref": "#/components/schemas/Cat_Type"
                },
                {
                    "type": "object",
                    "required": [
                        "pet_type"
                    ],
                    "properties": {
                        "bark": {
                            "type": "boolean"
                        },
                        "pet_type": {
                            "type": "string"
                        },
                        "breed": {
                            "type": "string",
                            "enum": [
                                "Dingo",
                                "Husky",
                                "Retriever",
                                "Shepherd"
                            ]
                        }
                    }
                }
            ]
        },
        "Cat_Type": {
            "type": "object",
            "required": [
                "pet_type"
            ],
            "properties": {
                "hunts": {
                    "type": "boolean"
                },
                "pet_type": {
                    "type": "string"
                },
                "age": {
                    "type": "integer"
                }
            }
        }
    }
}
```

oneOf with a Subschema Containing Multiple Mappings

You can use a `oneOf` pattern in which a subschema contains multiple mappings.

```
{
    "schemas": {
        "Pet": {
            "discriminator": {
                "propertyName": "pet_type",
                "mapping": {
                    "CAT": "#/components/schemas/Cat_Type",
                    "DOG": "#/components/schemas/Dog_Type",
                    "PUP": "#/components/schemas/Dog_Type"
                }
            },
            "oneOf": [
                {
                    "$ref": "#/components/schemas/Cat_Type"
                },
                {
                    "$ref": "#/components/schemas/Dog_Type"
                }
            ]
        },
        "Cat_Type": {
            "type": "object",
            "required": [
                "pet_type"
            ],
            "properties": {
                "hunts": {
                    "type": "boolean"
                },
                "pet_type": {
                    "type": "string"
                },
                "age": {
                    "type": "integer"
                }
            }
        },
        "Dog_Type": {
            "type": "object",
            "required": [
                "pet_type"
            ],
            "properties": {
                "bark": {
                    "type": "boolean"
                },
                "pet_type": {
                    "type": "string"
                },
                "breed": {
                    "type": "string",
                    "enum": [
                        "Dingo",
                        "Husky",
                        "Retriever",
                        "Shepherd"
                    ]
                }
            }
        }
    }
}
```

oneOf Defined as Items of an Array Type

You can use a `oneOf` pattern defined as items of an array type.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "properties": {
                    "Pets": {
                        "description": "The items in the collection.",
                        "items": {
                            "discriminator": {
                                "propertyName": "pet_type",
                                "mapping": {
                                    "CAT": "#/components/schemas/Cat_Type",
                                    "DOG": "#/components/schemas/Dog_Type"
                                }
                            },
                            "oneOf": [
                                {
                                    "$ref": "#/components/schemas/Cat_Type"
                                },
                                {
                                    "$ref": "#/components/schemas/Dog_Type"
                                }
                            ]
                        }
                    },
                    "title": "Pets",
                    "type": "array",
                    "x-cardinality": "1"
                }
            },
            "type": "object"
        }
    }
}
```

oneOf Defined as Items of a Top Level Array

You can use a `oneOf` pattern defined as items of a top level array.

```
{
    "components": {
        "schemas": {
            "Pets": {
                "description": "The items in the collection.",
                "items": {
                    "discriminator": {
                        "propertyName": "pet_type",
                        "mapping": {
                            "CAT": "#/components/schemas/Cat_Type",
                            "DOG": "#/components/schemas/Dog_Type"
                        }
                    },
                    "oneOf": [
                        {
                            "$ref": "#/components/schemas/Cat_Type"
                        },
                        {
                            "$ref": "#/components/schemas/Dog_Type"
                        }
                    ]
                },
                "title": "Pets",
                "type": "array",
                "x-cardinality": "1"
            }
        }
    }
}
```

oneOf with a Nested allOf Subschema Defined as $ref

You can use a `oneOf` pattern in which `allOf` is nested and defined as a reference (`$ref`).

```
{
    "schemas": {
```

```
"Animal": {
    "discriminator": {
        "propertyName": "pet_type",
        "mapping": {
            "CAT": "#/components/schemas/CatPet",
            "DOG": "#/components/schemas/DogPet"
        }
    },
    "oneOf": [
        {
            "$ref": "#/components/schemas/CatPet"
        },
        {
            "$ref": "#/components/schemas/DogPet"
        }
    ]
},
"CatPet": {
    "allOf": [
        {
            "$ref": "#/components/schemas/Pet"
        },
        {
            "$ref": "#/components/schemas/Cat"
        }
    ]
},
"DogPet": {
    "allOf": [
        {
            "$ref": "#/components/schemas/Pet"
        },
        {
            "$ref": "#/components/schemas/Dog"
        }
    ]
},
"Pet": {
    "type": "object",
    "required": [
        "pet_type"
    ],
    "properties": {
        "pet_type": {
            "type": "string"
        }
    }
},
"Dog": {
    "type": "object",
    "properties": {
        "bark": {
            "type": "boolean"
        },
        "breed": {
            "type": "string",
                    "enum": [
                        "Dingo",
                        "Husky",
                        "Retriever",
                        "Shepherd"
                    ]
                }
            }
        },
        "Cat": {
            "type": "object",
            "properties": {
                "hunts": {
                    "type": "boolean"
                },
                "age": {
                    "type": "integer"
                }
            }
        }
    }
}
```

oneOf with a Nested allOf Subschema Defined Inline

You can use a `oneOf` pattern in which the `allOf` is nested and defined inline.

```
{
    "schemas": {
        "Animal": {
            "discriminator": {
                "propertyName": "pet_type",
                "mapping": {
                    "DOG": "#/components/schemas/DogPet"
                }
            },
            "oneOf": [
                {
                    "allOf": [
                        {
                            "$ref": "#/components/schemas/Pet"
                        },
                        {
                            "$ref": "#/components/schemas/Cat"
                        }
                    ]
                },
                {
                    "$ref": "#/components/schemas/DogPet"
                }
            ]
        },
        "DogPet": {
            "allOf": [
                {
                    "$ref": "#/components/schemas/Pet"
                },
                {
                    "$ref": "#/components/schemas/Dog"
                }
            ]
        },
        "Pet": {
            "type": "object",
            "required": [
                "pet_type"
            ],
            "properties": {
                "pet_type": {
                    "type": "string"
                }
            }
        },
        "Dog": {
            "type": "object",
            "properties": {
                "bark": {
                    "type": "boolean"
                },
                "breed": {
                    "type": "string",
                    "enum": [
                        "Dingo",
                        "Husky",
                        "Retriever",
                        "Shepherd"
                    ]
                }
            }
        },
        "Cat": {
            "type": "object",
            "properties": {
                "hunts": {
                    "type": "boolean"
                },
                "age": {
                    "type": "integer"
                }
            }
        }
    }
}
```

oneOf with a Nested anyOf Subschema Defined as $ref

You can use a `oneOf` pattern in which a nested `anyOf` subschema is defined as a reference ($`ref`).

```
{
    "schemas": {
        "Animal": {
            "discriminator": {
                "propertyName": "pet_type",
                "mapping": {
                    "CAT": "#/components/schemas/CatPet",
                    "DOG": "#/components/schemas/DogPet"
                }
            },
            "oneOf": [
                {
                    "$ref": "#/components/schemas/CatPet"
                },
                {
                    "$ref": "#/components/schemas/DogPet"
                }
            ]
        },
        "CatPet": {
            "anyOf": [
                {
                    "$ref": "#/components/schemas/Pet"
                },
                {
                    "$ref": "#/components/schemas/Cat"
                }
            ]
        },
        "DogPet": {
            "anyOf": [
                {
                    "$ref": "#/components/schemas/Pet"
                },
                {
                    "$ref": "#/components/schemas/Dog"
                }
            ]
        },
        "Pet": {
            "type": "object",
            "required": [
                "pet_type"
            ],
            "properties": {
                "pet_type": {
                    "type": "string"
                }
            }
        },
        "Dog": {
            "type": "object",
            "properties": {
                "bark": {
                    "type": "boolean"
                },
                "breed": {
                    "type": "string",
                    "enum": [
                        "Dingo",
                        "Husky",
                        "Retriever",
                        "Shepherd"
                    ]
                }
            }
        },
        "Cat": {
            "type": "object",
            "properties": {
                "hunts": {
                    "type": "boolean"
                },
                "age": {
                    "type": "integer"
                }
            }
        }
    }
}
```

oneOf with a Nested anyOf Subschema Defined Inline

You can use a `oneOf` pattern in which a nested `anyof` subschema is defined inline.

```
 Collapse source
{
    "schemas": {
        "Animal": {
            "discriminator": {
                "propertyName": "pet_type",
                "mapping": {
                    "DOG": "#/components/schemas/DogPet"
                }
            },
            "oneOf": [
                {
                    "anyOf": [
                        {
                            "$ref": "#/components/schemas/Pet"
                        },
                        {
                            "$ref": "#/components/schemas/Cat"
                        }
                    ]
                },
                {
                    "$ref": "#/components/schemas/DogPet"
                }
            ]
        },
        "DogPet": {
            "allOf": [
                {
                    "$ref": "#/components/schemas/Pet"
                },
                {
                    "$ref": "#/components/schemas/Dog"
                }
            ]
        },
        "Pet": {
            "type": "object",
            "required": [
                "pet_type"
            ],
            "properties": {
                "pet_type": {
                    "type": "string"
                }
            }
        },
        "Dog": {
            "type": "object",
            "properties": {
                "bark": {
                    "type": "boolean"
                },
                "breed": {
                    "type": "string",
                    "enum": [
                        "Dingo",
                        "Husky",
                        "Retriever",
                        "Shepherd"
                    ]
                }
            }
        },
        "Cat": {
            "type": "object",
            "properties": {
                "hunts": {
                    "type": "boolean"
                },
                "age": {
                    "type": "integer"
                }
            }
        }
    }
}
```

anyOf Keyword Pattern Support

Use the `anyOf` keyword to ensure that the given data is valid against any of the subschemas.

> **Note:**
>
> The anyOf keyword pattern with a subschema being of simple type is not currently supported.
> Any other patterns not listed in the following sections should also be considered unsupported.

Oracle Integration supports the following anyOf keyword patterns.

- anyOf with All Subschemas Defined as $ref
- anyOf with All Subschemas Defined Inline
- anyOf with Subschemas Being a Mix of Inline and $ref
- anyOf Defined as Item of an Array Type
- anyOf Defined as Items of a Top Level Array
- anyOf with a Nested anyOf Subschema Defined as $ref
- anyOf with a Nested anyOf Subschema Defined Inline
- anyOf with a Nested allOf Subschema Defined as $ref
- anyOf with a Nested allOf Subschema Defined Inline
- anyOf with a Nested oneOf Subschema Defined as $ref
- anyOf with a Nested oneOf Subschema Defined Inline

anyOf with All Subschemas Defined as $ref

You can use an anyOf pattern in which all subschemas are defined as a reference ($ref).

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "$ref": "#/components/schemas/ItemProperties"
                    },
                    {
                        "$ref": "#/components/schemas/ItemProductionPrivateVO-
item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemProductionPrivateVO-
item-response-forChildren"
                    }
                ]
            }
        }
    }
}
```

anyOf with All Subschemas Defined Inline

You can use an `anyOf` pattern with all subschemas defined inline.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "required": [
                            "count",
                            "hasMore"
                        ],
                        "type": "object",
                        "properties": {
                            "totalResults": {
                                "type": "integer",
                                "format": "int32"
                            },
                            "count": {
                                "type": "integer",
                                "format": "int32"
                            },
                            "hasMore": {
                                "type": "boolean"
                            }
                        }
                    },
                    {
                        "type": "object",
                        "properties": {
                            "items": {
                                "title": "Items",
                                "type": "array",
                                "description": "The items in the collection.",
                                "items": {
                                    "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-response"
                                },
                                "x-cardinality": "1"
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

anyOf with Subschemas Being a Mix of Inline and $ref

You can use an `anyOf` pattern with subschemas being a mix of inline and references (`$ref`).

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "$ref": "#/components/schemas/CollectionProperties"
                    },
                    {
                        "type": "object",
                        "properties": {
                            "items": {
                                "title": "Items",
                                "type": "array",
                                "description": "The items in the collection.",
                                "items": {
                                    "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-response"
                                },
                                "x-cardinality": "1"
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

anyOf Defined as Item of an Array Type

You can use an `anyOf` pattern defined as an item of an array type.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "properties": {
                    "repeatingElement": {
                        "description": "The items in the collection.",
                        "items": {
                            "anyOf": [
                                {
                                    "$ref": "#/components/schemas/
ItemProperties"
                                },
                                {
                                    "$ref": "#/components/schemas/
ItemProductionPrivateVO-item"
                                }
                            ]
                        },
                        "title": "repeatingElement",
                        "type": "array",
                        "x-cardinality": "1"
                    },
                    "someStringElement": {
                        "type": "string"
                    }
                },
                "type": "object"
            }
        }
    }
}
```

anyOf Defined as Items of a Top Level Array

You can use an `anyOf` pattern defined as items of a top level array.

```
{
    "components": {
        "schemas": {
            "Pets": {
                "description": "The items in the collection.",
                "items": {
                    "anyOf": [
                        {
                            "$ref": "#/components/schemas/Cat_Type"
                        },
                        {
                            "$ref": "#/components/schemas/Dog_Type"
                        }
                    ]
                },
                "title": "Pets",
                "type": "array",
                "x-cardinality": "1"
            }
        }
    }
}
```

anyOf with a Nested anyOf Subschema Defined as $ref

You can use an `anyOf` pattern in which a nested `anyOf` subschema is defined as a reference (`$ref`).

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
patch-item"
```

```
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
item-patch-request-forChildren"
                    }
                ]
            },
            "ItemRootIccPrivateVO-patch-item": {
                "anyOf": [
                    {
                        "$ref": "#/components/schemas/ItemEffCategoryVO-patch-
item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
updatableFields"
                    }
                ],
                "title": "Item Extensible Flexfield"
            },
            "ItemRootIccPrivateVO-item-patch-request-forChildren": {
                "type": "object",
                "properties": {
                    "ItemEFFBItem__Details__EFFPrivateVO": {
                        "type": "array",
                        "items": {
                            "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-patch-request"
                        },
                        "x-cardinality": "1"
                    }
                }
            }
        }
    }
}
```

anyOf with a Nested anyOf Subschema Defined Inline

You can use an `anyOf` pattern in which a nested `anyOf` subschema is defined inline.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "anyOf": [
                            {
                                "$ref": "#/components/schemas/
ItemEffCategoryVO-patch-item"
                            },
                            {
                                "$ref": "#/components/schemas/
ItemRootIccPrivateVO-updatableFields"
                            }
                        ],
                        "title": "Item Extensible Flexfield"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
item-patch-request-forChildren"
                    }
                ]
            }
        }
    }
}
```

anyOf with a Nested allOf Subschema Defined as $ref

You can use an `anyOf` pattern in which a nested `allOf` subschema is defined as a reference (`$ref`).

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
patch-item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
item-patch-request-forChildren"
                    }
                ]
            },
            "ItemRootIccPrivateVO-patch-item": {
                "allOf": [
                    {
                        "$ref": "#/components/schemas/ItemEffCategoryVO-patch-
item"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
updatableFields"
                    }
                ],
                "title": "Item Extensible Flexfield"
            },
            "ItemRootIccPrivateVO-item-patch-request-forChildren": {
                "type": "object",
                "properties": {
                    "ItemEFFBItem__Details__EFFPrivateVO": {
                        "type": "array",
                        "items": {
                            "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-patch-request"
                        },
                        "x-cardinality": "1"
                    }
                }
            }
        }
    }
}
```

anyOf with a Nested allOf Subschema Defined Inline

You can use an `anyOf` pattern in which a nested `allOf` subschema is defined inline.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "allOf": [
                            {
                                "$ref": "#/components/schemas/
ItemEffCategoryVO-patch-item"
                            },
                            {
                                "$ref": "#/components/schemas/
ItemRootIccPrivateVO-updatableFields"
                            }
                        ],
                        "title": "Item Extensible Flexfield"
                    },
                    {
                        "$ref": "#/components/schemas/ItemRootIccPrivateVO-
item-patch-request-forChildren"
                    }
                ]
            }
        }
    }
}
```

anyOf with a Nested oneOf Subschema Defined as $ref

You can use an `anyOf` pattern in which a nested `oneOf` subschema is defined as a reference (`$ref`).

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "$ref": "#/components/schemas/ItemEffCategoryVO-patch-
item"
                    },
                    {
                        "type": "object",
                        "properties": {
                            "items": {
                                "title": "Items",
                                "type": "array",
                                "description": "The items in the collection.",
                                "items": {
                                    "$ref": "#/components/schemas/itemsV2-
ItemEffCategory-item-response"
                                },
                                "x-cardinality": "1"
                            }
                        }
                    }
                ]
            },
            "ItemEffCategoryVO-patch-item": {
                "discriminator": {
                    "propertyName": "CategoryCode",
                    "mapping": {
                        "Production": "schemas/ItemProductionPrivateVO-item-
response",
                        "ROOT_ICC": "schemas/ItemRootIccPrivateVO-item-
response"
                    }
                },
                "oneOf": [
                    {
                        "$ref": "schemas/ItemProductionPrivateVO-item-
response"
                    },
                    {
                        "$ref": "schemas/ItemRootIccPrivateVO-item-response"
                    }
                ]
            }
        }
    }
}
```

anyOf with a Nested oneOf Subschema Defined Inline

You can use an `anyOf` pattern in which a nested `oneOf` subschema is defined inline.

```
{
    "components": {
        "schemas": {
            "ItemRootIccPrivateVO-item-patch-request": {
                "anyOf": [
                    {
                        "$ref": "#/components/schemas/ItemProperties"
                    },
                    {
                        "discriminator": {
                            "propertyName": "CategoryCode",
                            "mapping": {
                                "Production": "#/components/schemas/
ItemProductionPrivateVO-item-response",
                                "ROOT_ICC": "#/components/schemas/
ItemRootIccPrivateVO-item-response"
                            }
                        },
                        "oneOf": [
                            {
                                "$ref": "#/components/schemas/
ItemProductionPrivateVO-item-response"
                            },
                            {
                                "$ref": "#/components/schemas/
ItemRootIccPrivateVO-item-response"
                            }
                        ]
                    }
                ]
            }
        }
    }
}
```

Support for OpenAPI Documents with External References

OpenAPI documents with external references are supported for both HTTP- and HTTPS-based URLs.

Note the following guidelines:

- The base URL for all references must be the same:

  ```
  http://dummyhost/resources/openapi/main.json
  http://dummyhost/resources/openapi/reference.json
  ```

  or

  ```
  https://dummyhost/resources/openapi/main.json
  https://dummyhost/resources/openapi/reference.json
  ```

- The protocol must be either HTTP or HTTPS for all references. (No redirections for HTTP to HTTPS or vice versa.)
- The URLs must be unprotected (that is, without authentication).
- The references are supported for both relative and absolute URLs.
  Examples of both references are as follows:

Absolute Reference	Relative Reference

"Request": {
    "$ref": "http://dummyhost/resources/openapi/RequestReference.json#/components/schemas/ReferredSchema"
}
	"Request": {
	    "$ref": "RequestReference.json#/components/schemas/ReferredSchema"
	}

- The references are supported only at the schema level (not supported at the path level).
- Only the paths associated in the main OpenAPI document are considered. Any paths in references are ignored.

Support for Publishing Interfaces for Oracle Integration Flows as OpenAPI Documents

Oracle Integration supports open standards for publishing integration flows and simplifying consumption of Oracle Integration flows from external systems.

In the same endeavor, the REST Adapter in Oracle Integration has started automatically publishing OpenAPI documents for all of the Oracle Integration flows that have the REST Adapter as a trigger connection. OpenAPI has become a de facto standard for describing a REST API. You can download the OpenAPI documents and use the documents to build the client applications for consuming the Oracle Integration flow exposed as a REST API.

Consumption of OpenAPI Multipart for JSON and Form Data

Request and response schemas for multipart (with a media type of multipart/form-data or multipart/mixed) are compliant with the OpenAPI 3.*x* standard. This enables you to consume the OpenAPI standard using the REST Adapter as a trigger connection in an integration.

For example, use cases such as the following are supported:

- Consuming an OpenAPI document with endpoints that consume or produce HTML form data
- Consuming an OpenAPI document with endpoints with a multipart/mixed content type
- Consuming an OpenAPI document with endpoints with multipart/form-data

See Consume and Publish OpenAPI Documents with Multipart/Mixed and Multipart/Form-Data.

Attachment Support

The following sections describe REST Adapter attachment capabilities in more detail.

Topics:

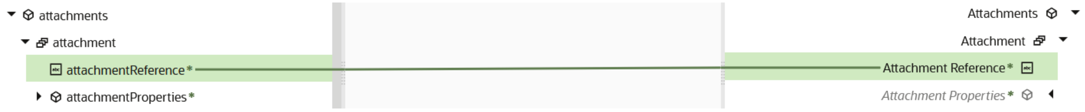
- Multipart Attachment Support for Trigger and Invoke Connections
- Support for application/octet-stream MIME Attachment (Binary) Payloads

Multipart Attachment Support for Trigger and Invoke Connections

The REST Adapter supports multipart attachments for trigger (inbound) and invoke (outbound) requests.

For example, you can send a review document attachment with the trigger (inbound) REST Adapter to an invoke (outbound) Adobe eSign or DocuSign for delivery to the downstream endpoint for signing.

If you want to send attachments from inbound to outbound (in request messages) or to download attachments from outbound to inbound (in response messages), then for each attachment you must map the **attachmentReference** from source to target in the mapper.



If you do not map **attachmentReference** in the mapper for a request, the outbound REST Adapter does not receive attachments from the inbound direction (multipart request). Similarly, if you do not map **attachmentReference** in the mapper for a response, the inbound REST Adapter does not receive attachments from the outbound REST Adapter (multipart response).

Understand the data structures of different types of configurations made using the REST Adapter or any application adapter exposing the REST API (used as a trigger) or consuming the REST API (used as an invoke).

There are two configuration categories of multipart request and response:

- A - Multipart/mixed or multipart/form-data configured with JSON or XML samples

  This configuration uses the attachments schema and payload schema. The payload schema is derived based on a sample JSON/XML schema provided during configuration in the Adapter Endpoint Configuration Wizard.

- B - Multipart/form-data with HTML form payload

  This configuration uses the attachments schema and a generic schema with a **ParameterList** element. The **ParameterList** element consists of an unbounded **parameter** element. Each **parameter** has a **name** attribute. The value of the parameter is set directly to the **parameter** element. If there are multiple parameters, the **parameter** element can be repeated in the mapper. The datatype of the **parameter** and name is string.

  > **Note:**
  >
  > This category is used when you select **Request is HTML Form** in the Request page of the Adapter Endpoint Configuration Wizard. This is similar for a response if you select **Response is HTML Form** in the Response page of the Adapter Endpoint Configuration Wizard.

Note the following details about both configuration categories:

- Attachments schema

The **attachments** element has an unbounded **attachment** element. This configuration supports receiving (on the source) or sending (on the target) multiple attachments. Each **attachment** element has **attachmentReference** and **attachmentProperties**.

- The **AttachmentReference** element contains the location where the attachment has been staged for access.

  The **AttachmentProperties** element provides metadata about a single attachment:

  - The **contentId** property sets the **Content-ID** header of the body part. The **Content-ID** header sets a unique ID for the body part.
  - The **contentType** property sets the **Content-Type** header of the body part. For example, if a PDF file is sent, the **contentType** property should be **application/pdf**. If the source is providing a multipart attachment, this is determined automatically. The mapper can set/override these values.
  - The **transferEncoding** property sets the **Content-Transfer-Encoding** header of the body part. This header's value is a single token specifying the type of encoding:

    ```
    Content-Transfer-Encoding := "BASE64" / "QUOTED-PRINTABLE" /
                                 "8BIT"   / "7BIT" /
                                 "BINARY" / x-token
    ```

    These values are not case sensitive. That is, **Base64**, **BASE64**, and **bAsE64** are all equivalent. An encoding type of **7BIT** requires that the body is already in a seven-bit, mail-ready representation. This is the default value (that is, **Content-Transfer-Encoding: 7BIT** is assumed if the **Content-Transfer-Encoding** header field is not present). See https://www.w3.org/Protocols/rfc1341/5_Content-Transfer-Encoding.html.
  - The **partName** property sets the **fileName** of the body part. The attached file/body part is saved by the target system with this name.
  - The **contentDisposition** property sets the **Content-Disposition** header of the body part.

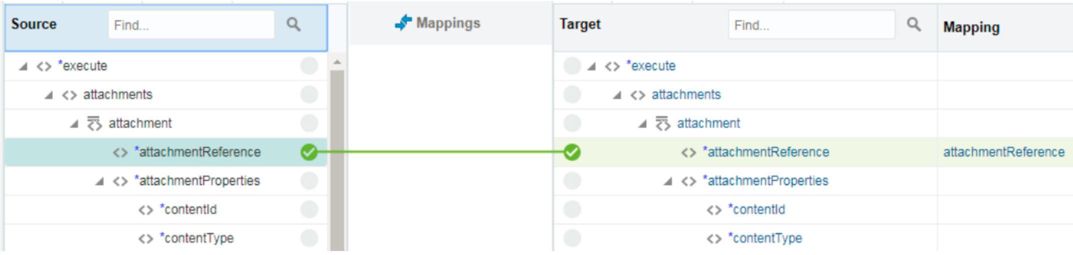
    In a multipart/form-data body, the HTTP **Content-Disposition** is a header to use on the subpart (that is, the attachment) of a multipart body to provide information about the field to which it applies. The **Content-Disposition** header value is generally set to **form-data**. The optional directive name and filename can also be used. For example:

    ```
    Content-Disposition: form-data
    Content-Disposition: form-data; name="fieldName"
    Content-Disposition: form-data; name="fieldName";
    filename="filename.jpg"
    ```
  - The **contentDescription** property sets some descriptive information with a given body part. For example, you can mark an **image** body as `a picture of the Space Shuttle Endeavor`. You can place such text in the **Content-Description** header field.
  - The **fileInputHtmlFieldName** property sets the name of the part from which the server must read the file.

Mapper configuration scenarios:

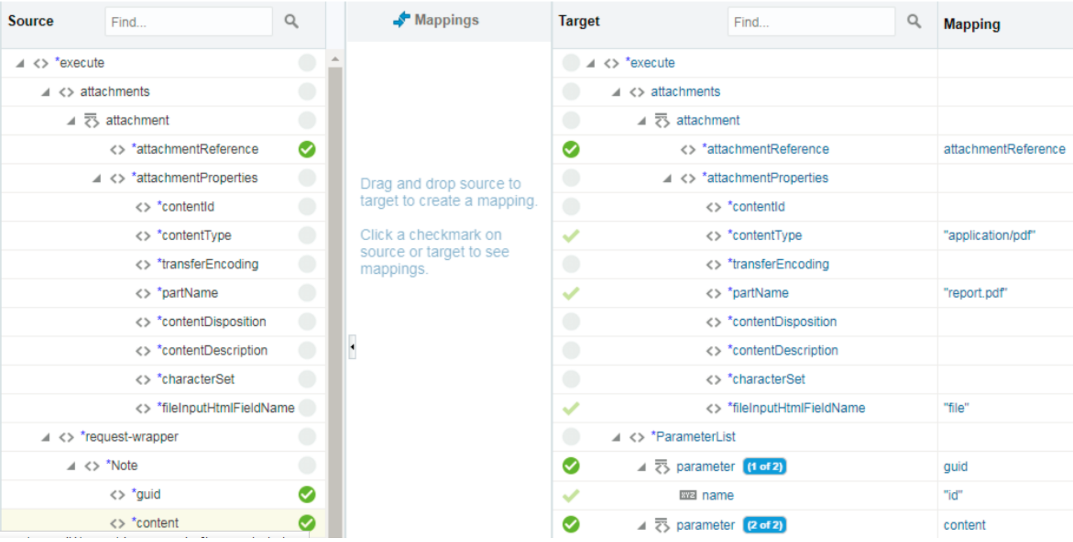
- Both source and target have multipart requests with JSON/XML payload (category A)

  The following sample map focuses only on the mapping of **attachmentReference** to the target. In this scenario, there is an assumption that only one attachment from the source is being mapped to the target. The mapping of the payload (request-wrapper node) between the source and target is not shown. You must perform that task.



- The source is multipart/mixed or multipart/form-data with JSON/XML payload (Category A). The target is multipart/form-data with form fields (Category B)

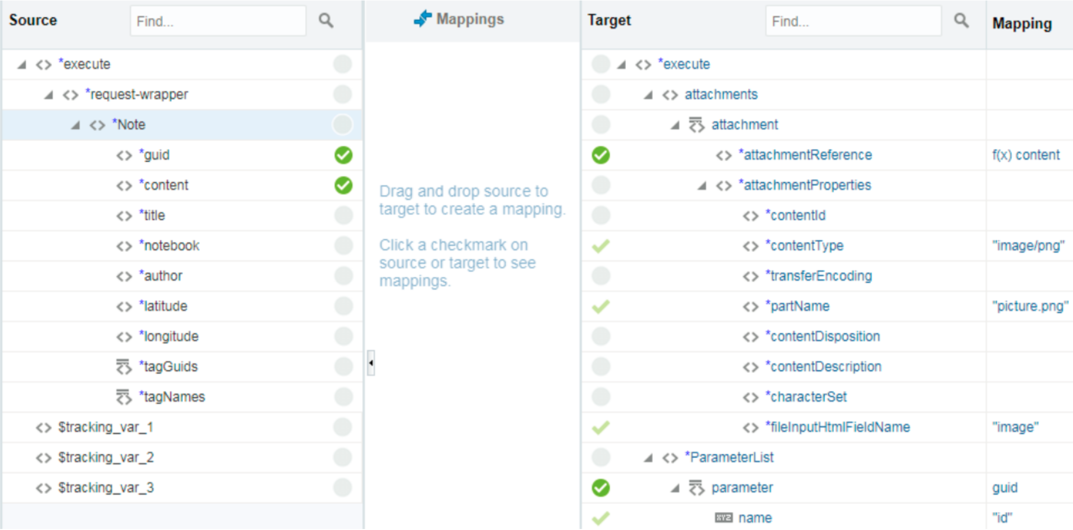
  The following map focuses on mapping of the attributes on the HTML form. There must be as many parameters in the **parameterList** as there are fields in the HTML form.



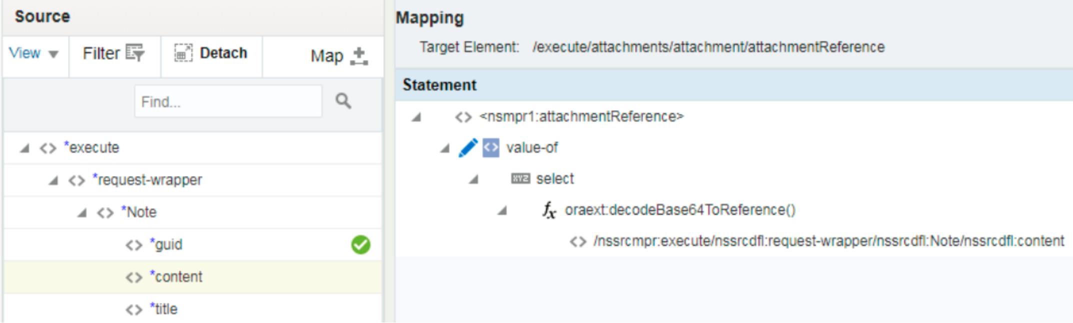
- Creating a reference from base64–encoded content. The source has a base64–encoded string and the target can be any of the three: multipart/mixed, multipart/form-data with JSON/XML payload, or multipart/form-data with HTML form payload.

  In the inbound payload, the **content** element is a base64–encoded string. This can be sent as an attachment in the outbound request.

  Since the inbound request is not multipart, but the outbound must be multipart, you must set multipart-specific properties in the mapper for the outbound direction. The **contentType** is set here to **image/png**, **partName** is set to **picture.png**, and **fileInputHtmlFieldName** is set to **image**. The assumption is that the target system is configured to read from a body part having **name="image"** in its content disposition. This is done with the element **fileInputHtmlFieldName**.



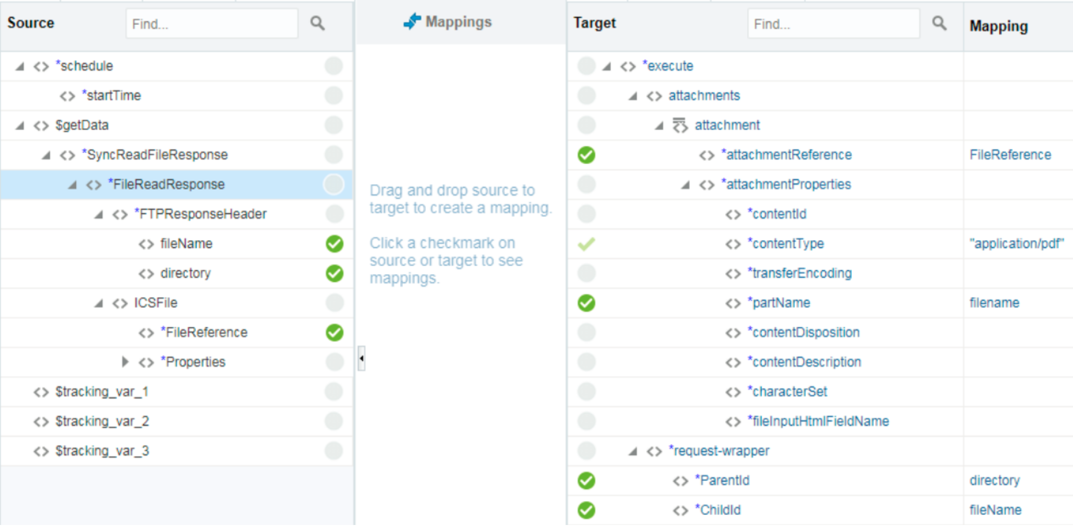
The base64 string can be converted into a reference using XSL function **decodeBase64ToReference** and the reference can be assigned to the **attachmentReference** element.



Target Element: /execute/attachments/attachment/attachmentReference

```
<nsmpr1:attachmentReference>
  value-of
    select
      oraext:decodeBase64ToReference()
        /nssrcmpr:execute/nssrcdfl:request-wrapper/nssrcdfl:Note/nssrcdfl:content
```

- The inbound is an FTP file read operation (nonmultipart) and the outbound is multipart/mixed with a JSON or XML payload.



Note:

- If the source is not multipart, but the target must be multipart, **contentType** and **partName** must be provided for the target through the mapper.
- The response mapper description is similar to the request mapper.

Several implementation patterns are provided. See Implement Common Patterns Using the REST Adapter.

Support for application/octet-stream MIME Attachment (Binary) Payloads

A MIME attachment with the content type `application/octet-stream` is a binary file. Typically, it is an application or a document that is opened in an application such as a spreadsheet or word processor. If the attachment has a filename extension associated with it, you may be able to determine what type of file it is.

For example, an `.exe` extension indicates a Windows or DOS program (executable), while a file ending in `.doc` is probably meant to be opened in Microsoft Word.

The `application/octet-stream` MIME type is used for unknown binary files. It preserves the file contents, but requires the receiver to determine file type, for example, from the filename extension. The Internet media type for an arbitrary byte stream is `application/octet-stream`.

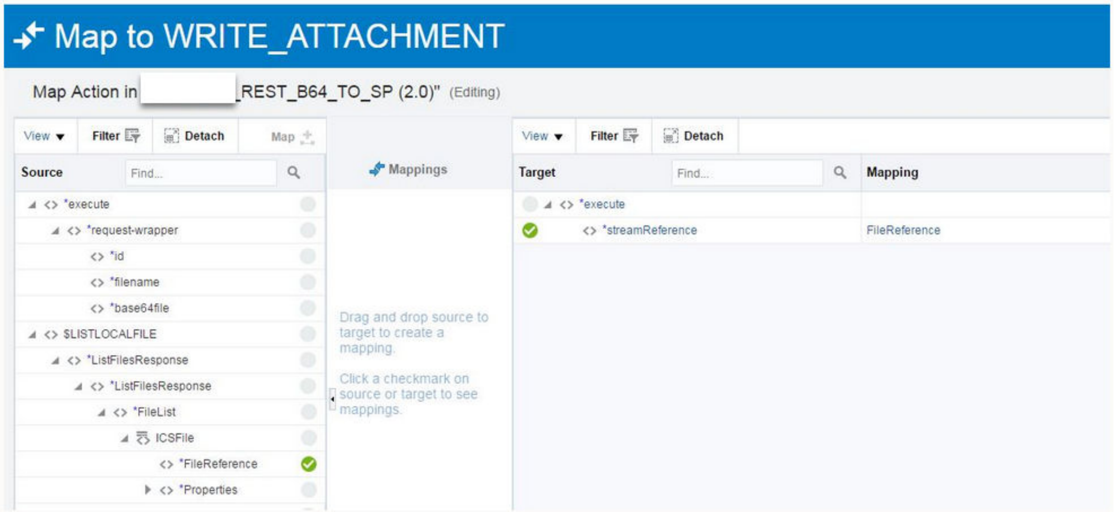
To use this feature, select the **Binary** option from the invoke Request/Response page when configuring the adapter as an invoke. When you select this option, you do not need to provide a schema because the payload has no structure.

This feature works with the `application/octet-stream` MIME type and any other type that can be sent as binary bytes. For example, the REST Adapter can send outbound requests or process outbound responses using the `application/pdf`, `application/zip`, `image/jpeg`, `image/png`, and other formats. Commonly used types shown in the dropdown are:

- `application/octet-stream`
- `application/pdf`
- `application/msword`
- `application/zip`
- `image/jpeg`
- `image/png`
- `image/bmp`
- `image/gif`

There is also a text box to provide a type not listed in the dropdown list (for example, `video/mp4` or `text/csv`).

The following screenshots show how binary payloads can be mapped.



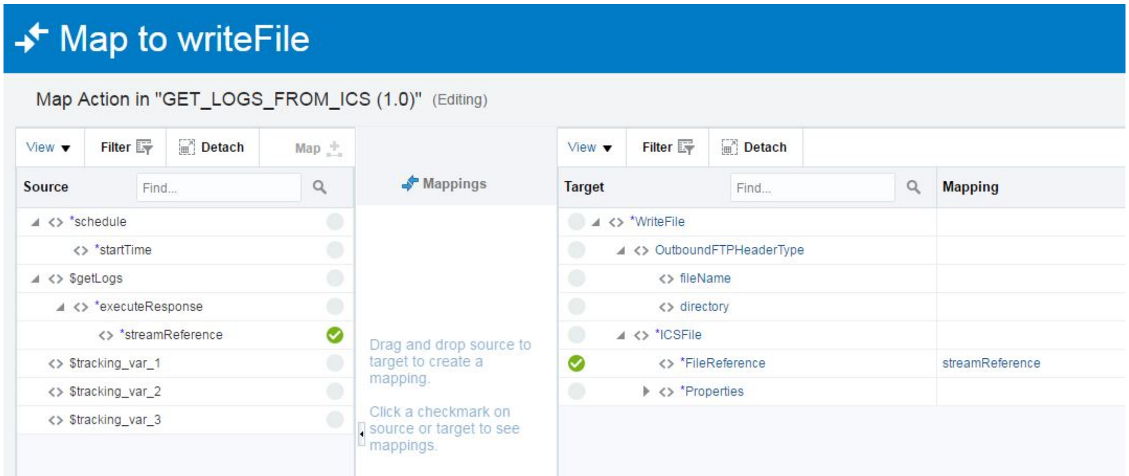
Map to WRITE_ATTACHMENT

Map Action in [redacted] REST_B64_TO_SP (2.0)" (Editing)

Drag and drop source to target to create a mapping.

Click a checkmark on source or target to see mappings.

Description of the illustration requestmappermime.png



Map to writeFile

Map Action in "GET_LOGS_FROM_ICS (1.0)" (Editing)

Drag and drop source to target to create a mapping.

Click a checkmark on source or target to see mappings.

Header, Token, Query Parameter, and Array Support

The following sections describe REST Adapter header, token, query parameter, and array capabilities in more detail.

Topics:

- Standard and Custom Header Support
- Nonstandard JWT Token Support
- RFC 3986 Support for Encoding Query Parameters
- Homogenous Multidimensional Array Support in JSON Documents
- Heterogeneous JSON Array Support

Standard and Custom Header Support

The REST Adapter supports standard and custom HTTP request and response headers in the invoke and trigger directions.

- Outbound (Invoke) direction

  HTTP headers enable you to use an outbound invocation to specify header properties. Many REST APIs expect certain properties to be specified in the HTTP headers (similar to SOAP APIs where you can specify header properties such as the WS address). Use the standard HTTP headers to specify these properties. You can also use the custom HTTP headers to specify properties. The REST APIs can expect the client application to pass properties in the custom headers, which can influence the behavior of the APIs. The standard and custom HTTP header properties configured in the Adapter Endpoint Configuration Wizard automatically start appearing in the mapper. You can map the header properties in the mapper.

- Inbound (trigger) direction

  You can expose integration flows as REST endpoints and enable client applications to populate the properties in the standard and custom headers. You can use these properties to create routing expressions in your integrations. The standard and custom HTTP header properties configured in the Adapter Endpoint Configuration Wizard automatically start appearing in the mapper. You can map the header properties in the mapper. See Create Routing Paths for Two Different Invoke Endpoints in Integrations and Design an Application Integration.

> **Note:**
>
> - If you want to send multiple values of a header, use comma separated values (CSVs). This is considered as one header and one value that consists of:
>
>   ```
>   val1 , val2 , val3 , ...
>   ```
>
>   The same value is propagated across the mapper and then to the outbound service. The outbound service must then interpret the CSVs of the header to be used as multiple values.
>
> - You can only set multiple values for custom headers. Setting multiple values for standard headers is *not* supported.
> - You cannot store multiple headers with the same name. The WSDL can only store one element with one unique name.

Nonstandard JWT Token Support

The use of nonstandard JWT tokens is supported. The JSON content type is a standard JWT token, while all other (for example, text or XML) are nonstandard JWT tokens. To fetch nonstandard JWT tokens from a REST service, use the following `regex` string.

- Use `regex` `“.*”` if the entire content is a JWT token. For this example, the entire content of the sample HTTP response is JWT token.

  ```
  HTTP/1.1 200 OK
  Date: Wed Jul 4 15:38:53 2012
  Connection: Keep-Alive:
  Content-Type: text/plain;charset=UTF-8
  Content-Length: 148
  MTgwNzE5NTY1NToxQkhzQlpaSXM0a21BV3NhbVBIclJOTFM4OGFxU09jNlRTdmFKSmczLVBqVHl
  WRF
  JwbWYxOFhmcnN6S0N6c3Fzb1JKbEh6U2IwSTdvflVuZWFXVjVmemhJNTJ1YVN6bFdDbTBG
  ```

- Use `regex` `“(?:.*?"my_token":")(.*?)(?:".*?)”`, if the JWT token is embedded inside a nonstandard response. For example, `my_token` is shown in the following sample HTTP response in which the JWT token is embedded inside a nonstandard response. This `regex` consists of a capturing group and noncapturing group. See https://www.regular-expressions.info/refcapture.html.

  ```
  HTTP/1.1 200 OK
  Date: Wed Jul 4 15:38:53 2012
  Connection: Keep-Alive:
  Content-Type: text/plain;charset=UTF-8
  Content-Length: 286
  "name":"raw-jwt"
  "my_token":"MTgwNzE5NTY1NToxQkhzQlpaSXM0a21BV3NhbVBIclJOTFM4OGFxU09jNlRTdmF
  KSm
  czLVBqVHlWRFJwbWYxOFhmcnN6S0N6c3Fzb1JKbEh6U2IwSTdvflVuZWFXVjVmemhJNTJ1YVN6b
  FdD
  bTBG"
  "id":"8353"
  ```

RFC 3986 Support for Encoding Query Parameters

The REST Adapter supports encoding query parameters.

The REST Adapter supports encoding query parameters in accordance with RFC 3986 standards. The default behavior is to encode the query parameters following the `application/x-www-form-urlencoded` scheme. For most older services that expect query parameters to be encoded following the `application/x-www-form-urlencoded` scheme, the default scheme should work. If you find the target endpoint not behaving correctly with the default encoding scheme, the REST Adapter can also be configured to strictly follow RFC 3986. A very common scenario in which the default behavior may not be desirable is when the target service expects space characters encoded as `%20` in the query parameters. In this case, the default behavior is to encode space characters as `+`. Some new services may also respond with `HTTP 400` (bad data) if query parameters are encoded in the `application/x-www-form-urlencoded` scheme. In these cases, you can switch to the RFC 3986 standard and check if the service responds correctly. To use RFC 3986 (and override the default behavior), perform the following steps to configure the REST Adapter as an invoke connection (and *not* as a trigger connection) in the Adapter Endpoint Configuration Wizard and in the mapper.

1. On the Basic Info page, select the **Custom** check box for **Configure Request Headers**.
2. On the Request Headers page, add the `x-ics-use-x-www-form-urlencoded` custom header and optionally provide a description.
3. Complete the Adapter Endpoint Configuration Wizard.
4. In the mapper, set the **x-ics-use-x-www-form-urlencoded** custom header to `false`.

The REST Adapter automatically encodes all query parameters in accordance with RFC 3986 in the outgoing request for this invoke connection.

Homogenous Multidimensional Array Support in JSON Documents

You can select a JSON sample with homogenous multidimensional arrays when configuring the REST Adapter in the Adapter Endpoint Configuration Wizard.

All JSON messages must be converted to XML before they can be processed by Oracle Integration at runtime. Semantically, there is no equivalent of multidimensional arrays in XML. To support multidimensional arrays, intermediate XML elements are generated that denote the beginning and ending of a nested array. When receiving a JSON message containing multidimensional arrays, these reserved elements are injected into the generated XML to denote the beginning and ending of a nested array. While converting XML elements back into JSON, the injected elements are converted into JSON with nested arrays.

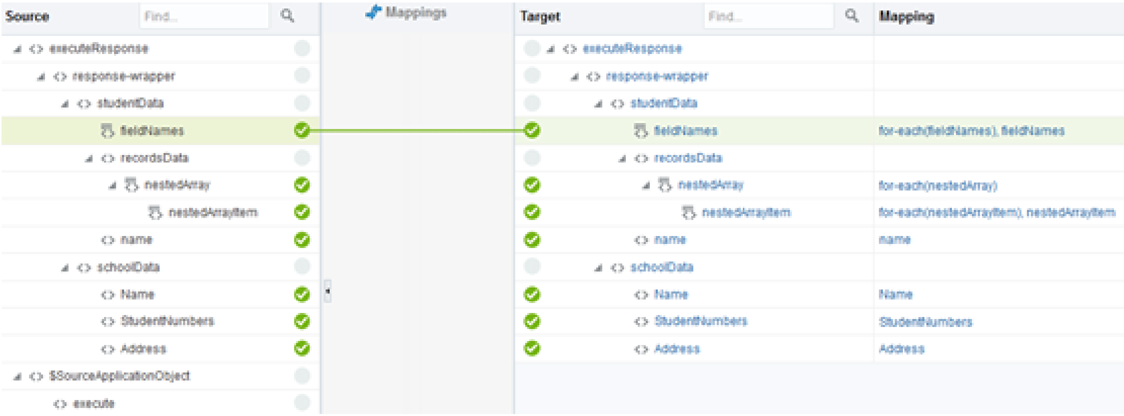
The following JSON document consists of a multidimensional array (`@ref "rercordsData”`).

```
{
    "studentData": {
        "fieldNames": [ "id","mobile_number" ],
        "recordsData": [ ["21","23"], ["+91123456789", "+91987654321" ]  ],
        "name": "jack"
    },
    "schoolData": {
        "Name": "ABCInternations",
        "StudentNumbers": 1300,
        "Address": "YYY streets Sector-44 India"
    }
}
```

The sample generated schema XML for the JSON document looks as follows:

```
<?xml version = '1.0' encoding = 'UTF-8'?>

<ns0:executeResponse xmlns:ns1="http://xmlns.oracle.com/cloud//REST/test/
types"
xmlns:ns0="http://xmlns.oracle.com/cloud//REST/test_REQUEST/types">

<ns1:response-wrapper>
  <ns1:studentData>
    <ns1:fieldNames>id</ns1:fieldNames>
    <ns1:fieldNames>mobile_number</ns1:fieldNames>
    <ns1:recordsData>
         <ns1:nestedArray>
                 <ns1:nestedArrayItem>21</ns1:nestedArrayItem>
                 <ns1:nestedArrayItem>23</ns1:nestedArrayItem>
         </ns1:nestedArray>
                         <ns1:nestedArray>
                                         <ns1:nestedArrayItem>+91123456789</
ns1:nestedArrayItem>
                                         <ns1:nestedArrayItem>+91987654321</
ns1:nestedArrayItem>
                         </ns1:nestedArray>
         </ns1:recordsData>
    <ns1:name>jack</ns1:name>
  </ns1:studentData>
  <ns1:schoolData>
        <ns1:Name>ABCInternations</ns1:Name>
        <ns1:StudentNumbers>1300</ns1:StudentNumbers>
        <ns1:Address>YYY streets Sector-44 India</ns1:Address>
   </ns1:schoolData>
</ns1:response-wrapper>
</ns0:executeResponse>
```

Elements in the nested array appear as `nestedArray` in the mapper and items in the elements appear as `nestedArrayItem`. You must map `nestedArray` as a `for-each` statement and `nestedArrayItem` as a `for-each` statement.



Heterogeneous JSON Array Support

In JSON, an array is an ordered collection of values that can be of various data types, such as strings, numbers, objects, arrays, booleans, or null. Homogeneous arrays are simply an ordered collection of values of the same type, whereas heterogeneous arrays are collections of disparate types.

Perform the following steps to configure a heterogeneous JSON array in the REST Adapter.

1. Provide sample JSON containing the JSON heterogeneous array in the REST Adapter. The sample must conform to the following rules:

   a. You must enrich the sample JSON payload with an additional metadata node `oic-json-metadata`. Provide the JSON path of the heterogeneous array in the sample and provide the JSON path to the discriminator.

   ```
   {
   "oic-json-metadata": {
   ```

```
  "heterogeneousArrays": [
    {
      "heterogeneousArrayPath": "$.entry",
      "heterogeneousArrayDiscriminatorPath": "$.entry.resourceType"
    }
  ]
}
```

b. Ensure the sample JSON is representative and adheres to the following rules:

   - All the types expected from the actual payload must be available in the sample.
   - Each discriminator must have at least one cell.
   - The first cell of a discriminator must be representative.
   - Each array type must have more than one cell.

2. Based on the metadata, Oracle Integration identifies the coordinates of heterogeneous arrays in this document and where to locate their discriminator.
   Every cell of the JSON payload having a heterogeneous array must contain the discriminator value.
3. For the heterogeneous array, an abstract type in the schema is generated. Each cell of the heterogeneous array is located. For each cell, the discriminator value is extracted and an extension type is created for the discriminator, which extends the abstract base type.
4. You must extend the base type for the required discriminators and map.
   The following types are not currently supported:
   - Nested heterogeneous arrays
   - Heterogeneous arrays at the top level
   - A nested heterogeneous array as an item of an array
   - Multiple heterogeneous arrays

Swagger Support

The following sections describe REST Adapter Swagger capabilities in more detail.

Topics:

- REST Endpoint Metadata and a Swagger Link to a REST Metadata Description

REST Endpoint Metadata and a Swagger Link to a REST Metadata Description

When you activate an integration with a REST Adapter trigger connection, an endpoint metadata URL link is provided at the top of the Integrations page.

For example:

```
integration Hello World (1.1.0) was activated successfully.
You can access it via http://host:port/ic/api/integration/v1/flows/rest/
HELLO_WORLD/1.0/metadata.
```

This link enables you to inspect the shape of the API. The metadata includes additional information about the endpoint description, the endpoint URI, and the Swagger URI.

Note the following details:

- If you import an IAR file with an endpoint description defined in the inbound (trigger) direction, update the connection, activate the integration, and access the metadata in a browser (for example, through a URL similar in structure to the following), the endpoint description is not shown even though the inbound direction has a description defined.

  ```
  http://host:port/ic/api/integration/v1/flows/rest/OLD_INTG_DESC/1.0/
  metadata
  ```

  This is expected behavior. The description is stored in a JCA file from which it is read and displayed. Existing integrations do not have this file. Even after upgrades, the existing integration does not show the endpoint description. To get the correct description, you must re-edit the REST Adapter to generate the artifacts again and re-activate the integration.
- If you attempt to re-edit an imported integration or existing integration in the Adapter Endpoint Configuration Wizard with a resource URI of `/metadata` or `/metadata/swagger`, you cannot navigate the wizard and receive an error. This is because the `/metadata` or `/metadata/swagger` keywords are reserved.
- If the relative URI has template parameters, then at runtime the value of the relative URI if resolved to `/metadata` or `/metadata/swagger` is treated as reserved for retrieving the integration metadata. Note the following behavior:
  - `/{param}`: Allowed - The integration cannot be invoked with the value of `param` as metadata and returns the metadata page.
  - `/{param}/swagger`: Allowed - The integration cannot be invoked with the value of `param` as metadata and returns the Swagger page.
  - `/metadata/{param}`: Allowed - The integration cannot be invoked with the value of `param` as Swagger and returns the Swagger page.
- Metadata and Swagger are only served depending on predefined reserve URIs for an integration. Resources with arbitrary URIs ending with values `metadata` or `swagger` are not confused with the endpoint documentation artifacts.

Mapper Connectivity Properties Support

The following section describes REST Adapter mapper connectivity property capabilities in more detail.

Topic:

Set REST Adapter Connectivity Properties in the Mapper

You can set REST Adapter connectivity properties in the mapper to propagate additional information to and from the target endpoint.

- Connectivity Properties (Invoke Response)

Restrictions

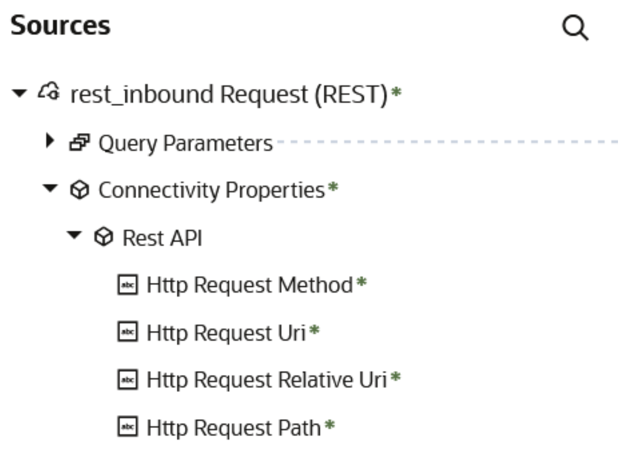
- You can customize the response status. However, this is not shown as part of the Swagger contract because runtime overrides are not known as part of the interface.
- The HTTP response status cannot be customized for the following conditions:
  - If the request is asynchronous one way, the response status is always `201`.
  - Errors occurring during trigger request/response handling are reported using predefined error codes.
  - Basic routing integrations don't allow fault handling. Error response in these scenarios cannot be customized.

Connectivity Properties (Trigger Request)

You can set the following properties for trigger requests in the mapper.

Property	Description
Http Request Method	This field contains the method name with which the REST Adapter trigger endpoint was invoked.
Http Request Uri	The absolute endpoint URI of the Oracle Integration REST Adapter trigger that was invoked by the client.
Http Request Relative Uri	The HTTP request relative URI. This is the relative URI of the Oracle Integration REST Adapter trigger REST endpoint.
Http Request Path	The HTTP request path. This is the path of the Oracle Integration REST trigger REST endpoint.

An asterisk in the mapper identifies headers and query and path parameters already sent along with the incoming request.

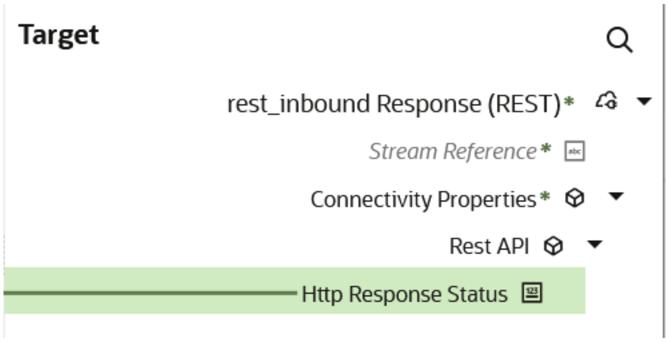


Connectivity Properties (Trigger Response)

You can set the following properties for trigger responses in the mapper.

Property	Description
Http Response Status	The value assigned to this property is sent back as the HTTP response status code by the Oracle Integration REST integration.

An asterisk in the mapper identifies headers/content that can be set with the outgoing response.



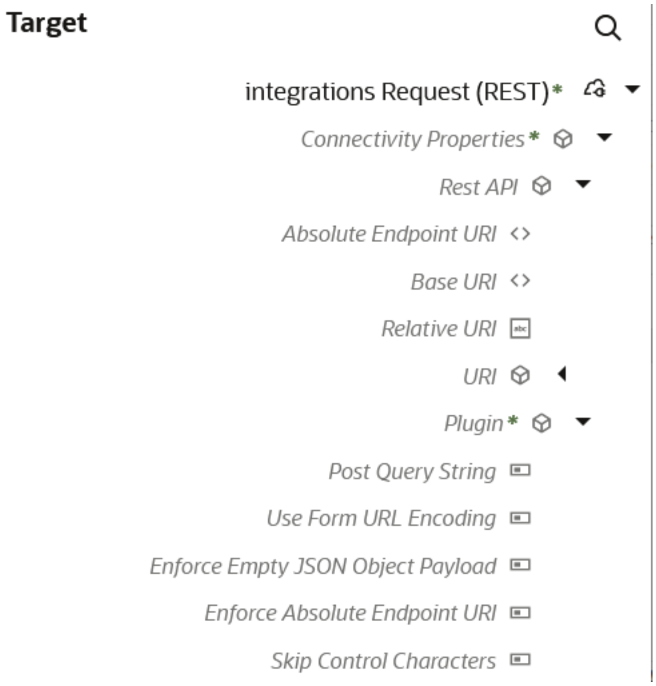
Connectivity Properties (Invoke Request)

You can set the following properties for invoke requests in the mapper.

Property	Description
Absolute Endpoint URI	When mapped to a valid URI, the request is routed to this URI.
Base URI	The base URI to which this request is routed. This is the equivalent of the base connection URL provided during connection configuration.
Relative URI	The relative URI of the resource. This must start with a `/`.
URI (Complex)	Use one of the following elements to substitute URI components. • scheme • host • port • path • query
Post Query String	When the runtime value is **true** and the HTTP verb is **POST**, the query string parameters are sent in the POST as form parameters. The default value is **false**.
Use Form URL Encoding	When the runtime value is **false**, the REST Adapter uses RFC–3986 compliant encoding to encode the query parameters. The default value is **true** (the equivalent of setting custom header **x-ics-use-x-www-form-urlencoded** to **false**). See RFC 3986 Support for Encoding Query Parameters.
Enforce Empty JSON Object Payload	When set to **true**, sets the body payload to an empty JSON object. The default value is **false**. See Send an Empty JSON Object.

Property	Description
Enforce Absolute Endpoint URI	Applies when you map an absolute endpoint URI to override the value configured in the REST Adapter. The enforcement of the absolute endpoint URI means that it does not encode or modify the URI when submitting it to the target.
Skip Control Characters	Use to skip control characters.

An asterisk in the mapper identifies headers and query and path parameters already sent along with the outgoing request.

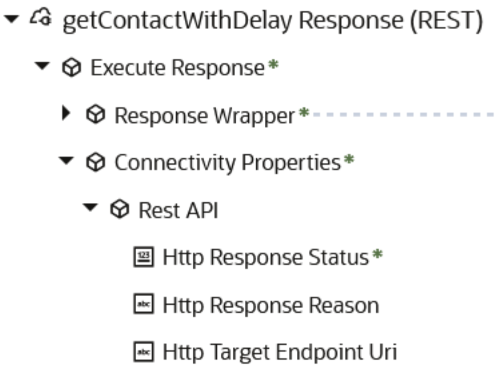


Connectivity Properties (Invoke Response)

You can set the following properties for invoke responses in the mapper.

Property	Description
Http Response Status	The HTTP response status returned by the target endpoint.
Http Response Reason	The reason corresponding to the HTTP response status returned by the target endpoint.
Http Target Endpoint Uri	The target endpoint that was invoked to receive this response.

An asterisk in the mapper identifies headers/content already sent as part of the incoming response received.



getContactWithDelay Response (REST)

Execute Response *

Response Wrapper *

Connectivity Properties *

Rest API

Http Response Status *

Http Response Reason

Http Target Endpoint Uri

REST Endpoint Support

The following sections describe REST endpoint capabilities in more detail.

Topics:

- Support for Dynamic REST Endpoints
- Configuration Parameters

Support for Dynamic REST Endpoints

The REST Adapter enables you to dynamically change the (invoke) outbound endpoint configuration. This feature is useful in the following scenarios:

- A REST endpoint is required to be invoked dynamically or an endpoint is not known at design time.
- Multiple REST services must be invoked, all of which accept the same input payload and return the same response payload as configured for the outbound endpoint. For such cases, this feature eliminates the need to create multiple connections for invoking each of these REST endpoints.

To change the endpoint configuration at runtime, you must provide a mapping for one or more of the various properties under **ConnectivityProperties**.

For example, the following steps describe how to configure an integration to invoke a REST endpoint determined at runtime:

1. Create and configure a REST Adapter as an invoke connection.
2. In the target pane of the mapper, expand **RestApi** under **ConnectivityProperties**. These elements are made available automatically through a static schema that is added to the user-provided schema.
3. Using the source schema in the source pane, create a mapping to **AbsoluteEndpointUri** in the target pane. Alternatively, you can also provide a static mapping. The REST Adapter uses the runtime value provided by this mapping to determine the REST endpoint to which to route this request.
4. You can similarly provide a source mapping to other target nodes under **ConnectivityProperties**. The REST Adapter uses the runtime values provided by these mappings to dynamically configure the request.
5. Activate and invoke the integration. The REST Adapter now invokes the endpoint URI determined at runtime.

6. Hover the mouse pointer over these properties in the mapper for a brief description. These descriptions are also provided below:

   - **AbsoluteEndpointUri**: Represents the absolute endpoint URL that the REST Adapter invokes. Empty values are ignored. To route the request to an endpoint URL determined at runtime, provide a mapping for this element. **AbsoluteEndpointUri** takes first precedence among other URL-related properties under **ConnectivityProperties**.
   - **BaseUri**: The equivalent of the base URL provided during connection configuration. To substitute only the base URI and keep the rest of the URL the same, provide a mapping for this element. The mapping is ignored if **AbsoluteEndpointUri** has a runtime value.
   - **RelativeUri**: Forms the part of the endpoint URI between **BaseUri** and **?**. This mapping has no impact if **BaseUri** has an empty runtime value or **AbsoluteEndpointUri** has a runtime value. The runtime value must start with a **/**.
   - **Uri**: Use the various elements under this node to substitute the specific parts with runtime values of an endpoint URL.
     - **Scheme**: Provide a mapping if you want to change only the scheme of the endpoint URL. The only supported values are **HTTP** and **HTTPS**.
     - **Host**: Provide a mapping if you want to change only the host of the endpoint URL.
     - **Port**: Provide a mapping if you want to change only the port of the endpoint URL.
     - **Query**: Provide a mapping if you want to change only the query portion of the endpoint URL. The query portion follows the **?**.
     - **Path**: Provide a mapping if you want to change only the path portion of the endpoint URL. A path is the part of a URI between the hostname and **?**.
   - **Plugin**: The various properties under this node impact the way the REST Adapter invokes the endpoint URL.
     - **PostQueryString**: When the runtime value is **true** and the HTTP verb is POST, the query string parameters are sent in the POST as form parameters. The default value is **false**.
     - **UseFormUrlEncoding**: When the runtime value is **false**, the REST Adapter uses RFC–3986 compliant encoding to encode the query parameters. The default value is **true** (the equivalent of setting custom header **x-ics-use-x-www-form-urlencoded** to **false**). See section “RFC 3986 Support for Encoding Query Parameters” for more information on **x-ics-use-x-www-form-urlencoded**. The **x-ics-use-x-www-form-urlencoded** custom header takes precedence when both properties are set.
     - **EnforceEmptyJSONObjectPayload**: When set to **true**, sets the body payload to an empty JSON object. The default value is **false**.

Note the following restrictions:

- The request and response schema must be the same as provided during configuration in the Adapter Endpoint Configuration Wizard.
- Template parameters are not supported while mapping these properties.
- An HTTP verb cannot be changed for the endpoint URL. For example, if the endpoint is configured to use POST, the outgoing request is a POST even if the endpoint URI changes at runtime.
- Since the endpoint URL is determined at runtime, there is no facility to test whether the security credentials provided during connection configuration also work with the new

endpoint URL. If the endpoint URL determined at runtime requires a different authorization header then the original URL, you may also have to provide a mapping for the authorization standard header.

Configuration Parameters

You configure the following parameters using the Adapter Endpoint Configuration Wizard to expose and consume a REST service:

- Relative resource path URI
- HTTP method (actions) to perform
- Template and query parameters
- Request/response message structure

Cross-Origin Resource Sharing (CORS) Support

The following section describes cross-origin resource sharing (CORS) capabilities in more detail.

Topics:

- Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS)

CORS defines a way in which a browser and server can interact to determine safely whether or not to allow the cross-origin request. CORS provides for more flexibility than same-origin requests, but is more secure than simply permitting all cross-origin requests.

Oracle Integration supports CORS in the REST Adapter trigger (inbound) direction. You configure CORS support in the Adapter Endpoint Configuration Wizard. See REST Adapter Trigger Resource Configuration Page and REST Adapter Trigger CORS Configuration Page.

CORS is supported by browsers based on the following layout engines:

- Blink- and Chromium-based browsers (Chrome 28, Opera 15, Amazon Silk, Android's 4.4+ WebView, and Qt's WebEngine).
- Gecko 1.9.1 (Firefox 3.5, SeaMonkey 2.0, and Camino 2.1) and above.
- MSHTML/Trident 6.0 (Internet Explorer 10) has native support. MSHTML/Trident 4.0 & 5.0 (Internet Explorer 8 & 9) provide partial support through the XDomainRequest object.
- Presto-based browsers (Opera) implement CORS as of Opera 12.00 and Opera Mobile 12, but not Opera Mini.
- WebKit (Safari 4 and above, Google Chrome 3 and above, possibly earlier).

The following browsers do not support CORS:

- Camino does not implement CORS in the 2.0.*x* release series because these versions are based on Gecko 1.9.0.
- As of version 0.10.2, Arora exposes WebKit's CORS-related APIs, but attempted cross-origin requests fail.[16].

For CORS to work, you must send an OPTIONS request. Using the `XMLHttpRequest` object in Javascript for (Ajax calls) automatically sends the OPTIONS request. If `XMLHttpRequest` is not used, then the OPTIONS request must be sent explicitly.

In the following example, an HTML client invokes an Oracle Integration CORS-based endpoint using `XMLHttpRequest`.

```
<html>

<script language="javascript">

var invocation = new XMLHttpRequest();
var url =
"<ics endpoint url>";
// Use postman to generate authCode. Sample is provided below
var authCode = 'Basic <base64encoded authorization string>';

function callOtherDomain(){   if(invocation)     {
invocation.open('GET', url, true);
invocation.setRequestHeader('Accept', 'application/json');
invocation.setRequestHeader('X-Cache','aaa');
invocation.setRequestHeader('X-Forwarded-For','fwd1');
invocation.setRequestHeader('Authorization',authCode);
invocation.onreadystatechange = stateChangeEventHandler;
invocation.send();
}
}

function stateChangeEventHandler()
{
//  check whether the data is loaded
if (invocation.readyState==4)
 {  // check whether the status is ok
  if (invocation.status==200)   {
  //alert(invocation.responseText)
document.getElementById("myTextarea").value = invocation.responseText
document.write("hello");
document.write(invocation.responseText);
  }
  else
  {
    alert ("Error Occurred")
  }
   }
}

</script>
<body onload="callOtherDomain()">
<br><br>
<textarea id="myTextarea" name="mytextarea1"></textarea><br><br>
</body>
</html>
```

Some browsers may also have security restrictions such as the same origin policy or a similar name that prevents using CORS. For example, to access a CORS-enabled endpoint using a Chrome browser, you may have to start it with web security disabled as follows.

```
chrome.exe --user-data-dir="C:/Chrome dev session" --disable-web-security
```

Complex Schema Support

The following section describes REST Adapter complex schema capabilities in more detail.

Topics:

- Complex Schema Support

Complex Schema Support

Support is provided for XSDs that can import and include other XSDs. The included XSDs in the ZIP file can import the XSD from an HTTP location. All XSD files must be added to a ZIP file and uploaded when configuring the REST Adapter in the Adapter Endpoint Configuration Wizard.

In the following example, the hierarchy of the ZIP file to upload is as follows:

```
zipxsd.zip
  first.xsd
  second (folder)
    second.xsd
```

`first.xsd` imports `second.xsd`.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://xmlns.oracle.com/first"
targetNamespace="http://xmlns.oracle.com/first"
      xmlns:tns1="http://xmlns.oracle.com/second">
<xs:import schemaLocation="./second/second.xsd"
targetNamespace="http://xmlns.oracle.com/second"/>
<xs:import schemaLocation="https://example.com/fscmService/ItemServiceV2?
XSD=/xml/datagraph.xsd" targetNamespace="commonj.sdo"/>
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string"/>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="tns1:author"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The contents of `second.xsd` are as follows.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://xmlns.oracle.com/second"
```

```
targetNamespace="http://xmlns.oracle.com/second">
<xs:import schemaLocation="https://example.com/fscmService/ItemServiceV2?
XSD=/mycompany/apps/scm/productModel/items/itemServiceV2/ItemAttachment.xsd"
targetNamespace="http://xmlns.oracle.com/apps/scm/productModel/items/
itemServiceV2/"/>
<xs:complexType name="author">
    <xs:sequence>
       <xs:element name="name" type="xs:string"/>
       <xs:element name="address" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="Admin">
    <xs:complexType>
          <xs:sequence>
             <xs:element name="AdminName" type="xs:string"/>
             <xs:element name="AdminAdd" type="xs:string"/>
          </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

> **Note:**
>
> If you are importing from HTTPS locations, ensure that you import the SSL certificates into Oracle Integration.

Resource Principal Session Token Support

The REST Adapter supports the Resource Principal Session Token (RPST). RPST enables an Oracle Integration instance (the resource) to authenticate itself with and consume other Oracle Cloud Infrastructure services, such as Oracle Cloud Infrastructure Functions, Oracle Cloud Infrastructure Object Storage, Oracle Cloud Infrastructure Vision, and more.

To use RPST, you create a dynamic group and specify a policy in the Oracle Cloud Infrastructure Console to enable access to Oracle Cloud Infrastructure services. Once these prerequisites are completed, you simply select the OIC Service Invocation security policy when configuring the REST Adapter as an invoke connection on the Connections page. No additional user configuration is required.

See RPST and OCI Service Invocation Security Policy Use.

The RPST authentication process works as follows:

1. You create a policy to grant the Oracle Integration instance access to a specific Oracle Cloud Infrastructure service or to all Oracle Cloud Infrastructure services in a specific compartment.
2. The OAuth client credentials flow obtains an identity domain bearer token that represents the Oracle Integration instance.
3. The identity domain bearer token calls a region-specific token exchange API to exchange the bearer token for an RPST token. The RPST token is only valid for resources to which the dynamic group has been granted access by the policy. The token is valid for one hour.
4. The RPST token is used to sign the request to the Oracle Cloud Infrastructure services specified in the policy (`keyId=`*`rpst_token`*).

A high-level overview of an integration that calls an Oracle Cloud Infrastructure service is provided. See Access Oracle Cloud Infrastructure Service Resources Using RPST.

JWT Assertion Support for Outbound Invocations

You may have a business need to invoke a service provider that does not regard an OAuth client secret as secure. For these scenarios, JWT assertions can be used. JWT assertions supplement all flavors of OAuth by authenticating the client application without the use of a client secret.

- Capabilities
- JWT Assertion Authentication Process

Capabilities

Trust is established with a key pair exchange instead of a client secret. No client secrets are shared. The National Health Service (NHS) and Fast Healthcare Interoperability Resources (FHIR) are examples of service providers that have moved away from client secret authentication to JWT assertions. The REST Adapter supports both JWT client and user assertions in the outbound (invoke) direction with the following security policies on the Connections page:

- OAuth Client Credentials using JWT Client Assertion
- OAuth using JWT User Assertion

JWT assertions provide the following capabilities:

- Full header and body claims control
- Multiple algorithm support (such as RSA)
- Full customization of form-data payloads (with an option to send the client secret, if necessary). Some providers don't follow standards to get the access token and prefer to customize their payloads.
- Session support (refresh token caching and transient access token handling)
- User-access token assertion support
- Support for the different implementations of JWT provided by the following services:
  - NHS
  - FHIR
  - DocuSign
  - Adobe eSign
  - Microsoft
  - Okta

JWT Assertion Authentication Process

The JWT assertion authentication process works as follows:

1. You manually create and upload a private signing key on the Certificates page in Oracle Integration.
2. You provide JWT header and payload files on the Connections page to formulate the JWT assertion, including entering the same private signing key name you specified on the Certificates page.

3. Oracle Integration uses the private signing key name to generate the JWT assertion.
4. The JWT assertion is used to call the access token URI to obtain the access token from the service provider.
5. The access token is used to call the REST API of the service provider.

A high-level use case is provided that describes how to create an integration with JWT assertion support. See Invoke a Service Provider API with a JWT Assertion.

Watch a video to learn more:

Video

Data Obfuscation Support

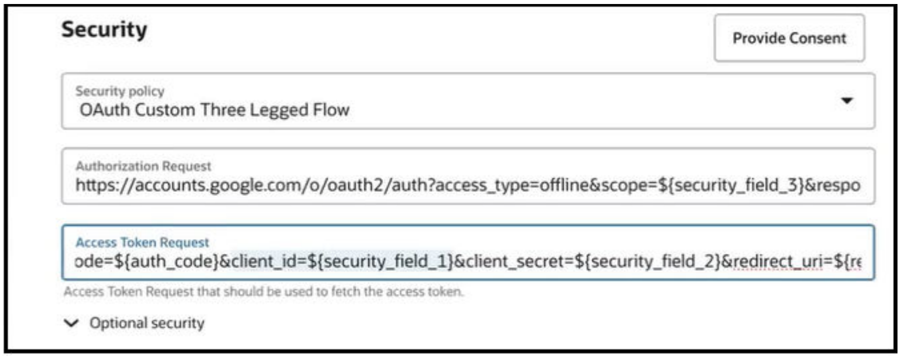
The REST Adapter supports the obfuscation of sensitive data, such as client ID, client secret, scopes, API key, and others when configuring the **Authorization Request**, **Access Token Request**, and **Refresh Token Request** fields for the OAuth Custom Three Legged Flow security policy and the **Access Token Request** and **Refresh Token Request** fields for the OAuth Custom Two Legged Flow security policy.

When creating the `curl` commands for these security policies on the Connections page, you may have data that you don't want to expose in clear text. For example, you may want to hide the `client_secret` value exposed in the `curl` command.

```
&client_secret=xxxasdfasdfadasdfasdfasdfasdfas
```

To hide this data, you can assign variables to your `curl` command parameters, such as `&scope=${security_field_3}`, `&client_secret=${security_field_2}`, `&client_id=${security_field_1}`, and others.

The following example shows the OAuth Custom Three Legged Flow security policy with assigned variables. You can follow a similar convention if using the OAuth Custom Two Legged Flow security policy.



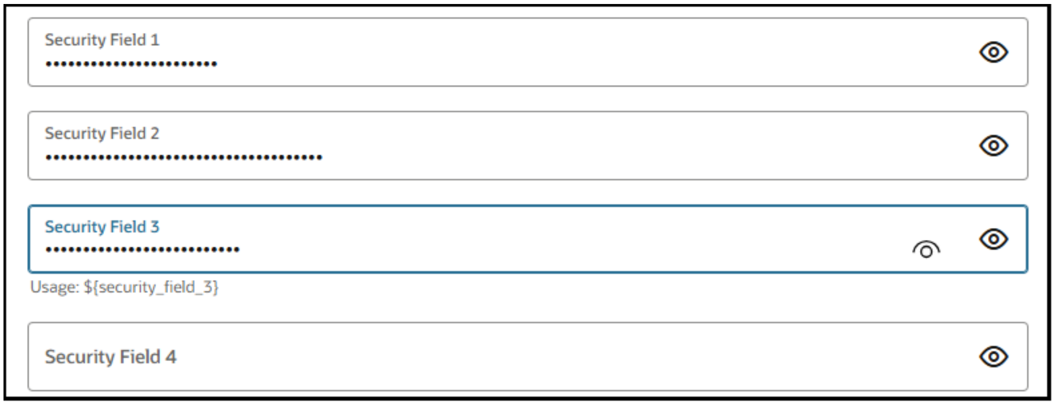
Security policy: OAuth Custom Three Legged Flow

Authorization Request: https://accounts.google.com/o/oauth2/auth?access_type=offline&scope=${security_field_3}&respo

Access Token Request: ode=${auth_code}&client_id=${security_field_1}&client_secret=${security_field_2}&redirect_uri=${re

Access Token Request that should be used to fetch the access token.

You then expand the **Optional security** section, and enter the actual values for these variables in obfuscated format.



Security Field 1

Security Field 2

Security Field 3

Usage: ${security_field_3}

Security Field 4

Up to four security fields are supported:

- `${security_field_1}`
- `${security_field_2}`
- `${security_field_3}`
- `${security_field_4}`

Data obfuscation is an optional feature. You can use variables where you want and leave other parameter values in clear text format. You can also continue to leave all data in clear text format. The choice is yours. See Configure Connection Security.

Use Case 1

This use case shows how to obfuscate the `api_key` value with `security_field_2` and `client_secret` value with `security_field_1`.

Without Security Fields	With Security Fields
Access token refresh: -X POST https://auth.company.com/oauth/token?grant_type=authorization_code&code=${auth_code}&client_id=1234556789**&api_key=12345654321&client_secret=xxxasdfasdfadasdfasdfasdfasdfas**&redirect_uri=${redirect_uri}	Access token refresh: -X POST https://auth.company.com/oauth/token?grant_type=authorization_code&code=${auth_code}&client_id=1234556789**&api_key=${security_field_2}&client_secret=${security_field_1}**&redirect_uri=${redirect_uri}
Refresh token request: -X POST https://auth.company.com/oauth/token?grant_type=refresh_token&refresh_token=${refresh_token}&client_id=1234556789**&client_secret=xxxasdfasdfadasdfasdfasdfasdfas**	Refresh token request: -X POST https://auth.company.com/oauth/token?grant_type=refresh_token&refresh_token=${refresh_token}&client_id=1234556789**&client_secret=${security_field_1**}

Use Case 2

This use case also shows how to obfuscate the api_key with security_field_2 and client_secret with security_field_1.

Without Security Fields	With Security Fields
Access token request: -X POST https://auth.company.com/oauth/token?grant_type=authorization_code&code=${auth_code}&client_id=1234556789**&api_key=12345654321&client_secret=xxxasdfasdfadasdfasdfasdfasdfas**&redirect_uri=${redirect_uri}	Access token request: -X POST https://auth.company.com/oauth/token?grant_type=authorization_code&code=${auth_code}&client_id=1234556789**&api_key=${security_field_2}&client_secret=${security_field_1}**&redirect_uri=${redirect_uri}
Refresh token request: -X POST https://auth.bullhornstaffing.com/oauth/token?grant_type=refresh_token&refresh_token=${refresh_token}**&api_key=12345654321&client_id=1234556789&client_secret=xxxasdfasdfadasdfasdfasdfasdfas**	Refresh token request: -X POST https://auth.bullhornstaffing.com/oauth/token?grant_type=refresh_token&refresh_token=${refresh_token}&client_id=1234556789**&api_key=${security_field_2}&client_secret=${security_field_1}**

Use Case 3

This use case also shows how to obfuscate the api_key with security_field_2 the client_secret with security_field_1.

Without Security Fields	With Security Fields
Access token request: -X -H 'account_password:${security_field_3}' POST https://auth.company.com/oauth/token?grant_type=authorization_code&code=${auth_code}&client_id=1234556789**&api_key=12345654321&client_secret=xxxasdfasdfadasdfasdfasdfasdfas**&redirect_uri=${redirect_uri}	Access token request: -X POST https://auth.company.com/oauth/token?grant_type=authorization_code&code=${auth_code}&client_id=1234556789**&api_key=${security_field_2}&client_secret=${security_field_1}**&redirect_uri=${redirect_uri}

3 Create a REST Adapter Connection

A connection is based on an adapter. You define connections to the specific cloud applications that you want to integrate.

Topics:

Prerequisites for Creating a Connection

You must satisfy the following prerequisites to create a connection with the REST Adapter:

- OAuth Security Policies
- SSL Endpoints
- Amazon Web Services (AWS) REST API Consumption
- OCI Signature Version 1 Security Policy Use
- RPST and OCI Service Invocation Security Policy Use
- JWT Assertions Outbound Use
- Prerequisites for Identity Propagation Between the REST Adapter and Oracle NetSuite

OAuth Security Policies Use

If you are using one of the OAuth security policies, you must already have registered your client application to complete the necessary fields on the Connections page. The Basic Authentication and No Security Policy security policies are exempted.

Before a client application can request access to resources on a resource server, the client application must first register with the authorization server associated with the resource server.

The registration is typically a one-time task. Once registered, the registration remains valid, unless the client application registration is revoked.

At registration time, the client application is assigned a client ID and a client secret (password) by the authorization server. The client ID and secret are unique to the client application on that authorization server. If a client application registers with multiple authorization servers (for example, Facebook, Twitter, and Google), each authorization server issues its own unique client ID to the client application.

```
@ref: http://tutorials.jenkov.com/oauth2/authorization.html
```

For OAuth configuration, read the provider documentation carefully and provide the relevant values.

SSL Endpoints Use

For SSL endpoints, obtain and upload a server certificate. See Upload a Certificate to Connect with External Services.

Amazon Web Services (AWS) REST API Consumption

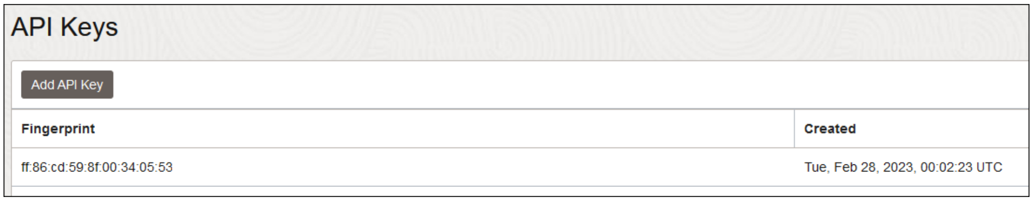
Before you can create a connection that consumes an Amazon Web Services (AWS) REST API, you must obtain the necessary access and secret keys. See Understanding and Getting Your Security Credentials.

OCI Signature Version 1 Security Policy Use

To configure the REST Adapter to use the OCI Signature Version 1 security policy on the Connections page, you must perform several tasks:

- Create an API signing key. You then specify the signing key in Oracle Cloud Infrastructure.
  1. Create the signing key in Oracle Cloud Infrastructure using `openssl`. The key must be in RSA (PKCS1) format. During creation, you also create a pass phase to protect the key. Both the key and pass phrase are required when configuring the OCI Signature Version 1 security policy on the Connections page. See Creating a Key Pair.
     If the key downloaded from the Oracle Cloud Infrastructure Console is in PKCS8 format, it must be converted to RSA (PKCS1) format. See Convert a Private Key from PKCS8 to RSA (PKCS1) Format for the OCI Signature Version 1 Security Policy.

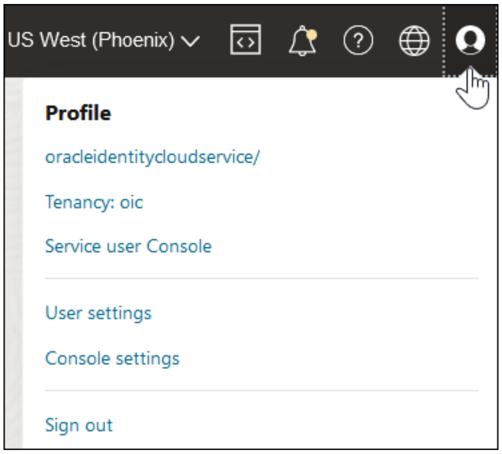
     Existing connections already using the OCI Signature Version 1 security policy do not need to be upgraded because they continue to work.
  2. Sign in to the Oracle Cloud Infrastructure Console.
  3. Open the navigation menu and click **Identity & Security**. Under **Identity**, click **Users**. A list of users in your tenancy is displayed.
  4. On the Users page, click the link of the user name to use.
  5. Scroll down to the **API Keys** section, and click **Add API Key**.
  6. In the Add API Key dialog, enter the contents of the public key you created, and click **Add**.
  7. Copy the finger print value generated by Oracle Cloud Infrastructure. You need this value when configuring the OCI Signature Version 1 security policy on the Connections page.



Fingerprint	Created
ff:86:cd:59:8f:00:34:05:53	Tue, Feb 28, 2023, 00:02:23 UTC

- Obtain the tenancy OCID and user OCID details in the Oracle Cloud Infrastructure Console. When you sign up for Oracle Cloud Infrastructure, Oracle creates a tenancy for your company, which is a secure and isolated partition within Oracle Cloud Infrastructure where you can create, organize, and administer your cloud resources.
  1. Sign in to the Oracle Cloud Infrastructure Console.
  2. Open the navigation menu and click **Governance & Administration**. Under **Account Management**, click **Tenancy Details**.

3. In the **Tenancy information** section, click **Show** to display the **OCID** tenancy value.
4. Copy the value. You need this value when configuring the OCI Signature Version 1 security policy on the Connections page.
5. In the header, click the **Profile** icon and select **User Settings**.



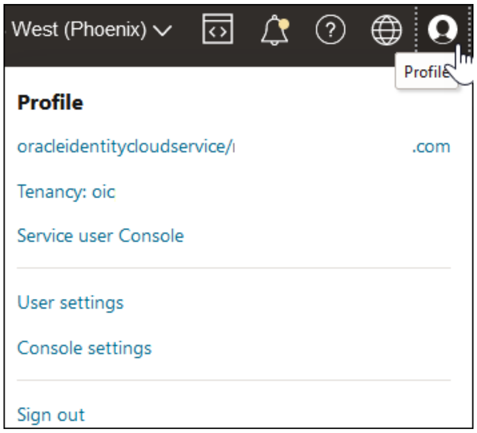
> **Note:**
>
> You can also open the navigation menu and click **Identity & Security**, then under **Identity**, click **Users** to access the user profile.

6. Click **Show** to display the **OCID** user value.
7. Copy the value. You need this value when configuring the OCI Signature Version 1 security policy on the Connections page.

RPST and OCI Service Invocation Security Policy Use

To use RPST and the OCI Service Invocation security policy, you must satisfy the following prerequisites.

- Ensure that your cloud tenancy uses identity domains. The OCI Service Invocation security policy that you select on the Connections page does *not* work in cloud tenancies that are not enabled for identity domains. If you are unsure, ask your administrator for details. Your cloud tenancy does *not* use identity domains if you observe either of the following:
  - In the upper right corner, you select your **Profiles** icon and don't see an entry for identity domain.



Profile

oracleidentitycloudservice/ .com

Tenancy: oic

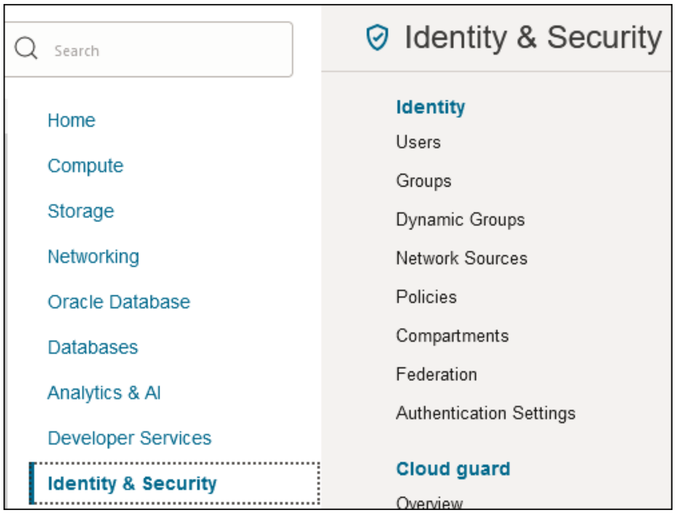
Service user Console

User settings

Console settings

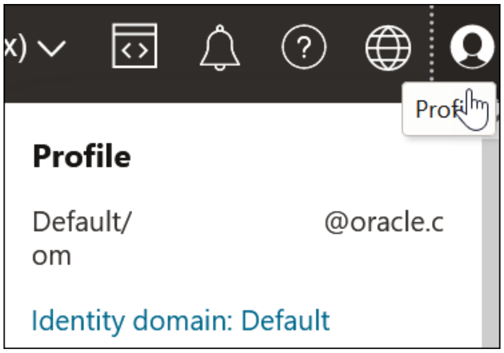
Sign out

- From the ☰ menu, you select **Identity & Security** and don't see **Domains** under the **Identity** section.

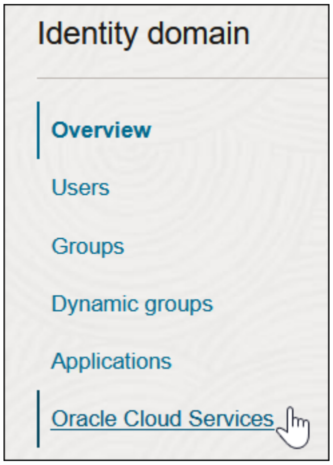


- Create the required dynamic group and assign a policy to that group to allow your Oracle Integration instance to access Oracle Cloud Infrastructure services, such as Oracle Cloud Infrastructure Functions, Oracle Cloud Infrastructure Object Storage, Oracle Cloud Infrastructure Vision, and more. The policy defines the permissions for the dynamic group and determines which operations the dynamic group can perform on Oracle Cloud Infrastructure services.

  1. Log in to the Oracle Cloud Infrastructure Console.
  2. Obtain the client ID of the OAuth application for the Oracle Integration instance.
     a. In the upper right corner, select **Profile**, then click the identity domain.



b. In the left navigation pane, click **Oracle Cloud Services**.

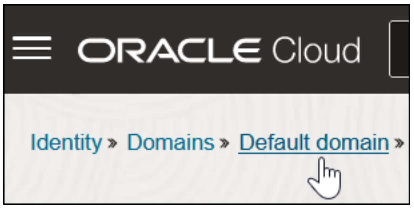


The Oracle Cloud Services page for your domain appears.

c. In the **Name** column, click your service instance.

d. Scroll down to the **General Information** section and copy the client ID value to use to create your dynamic group.

3. Scroll to the breadcrumbs at the top and click **Default domain**.



4. In the left navigation pane, click **Dynamic groups**.

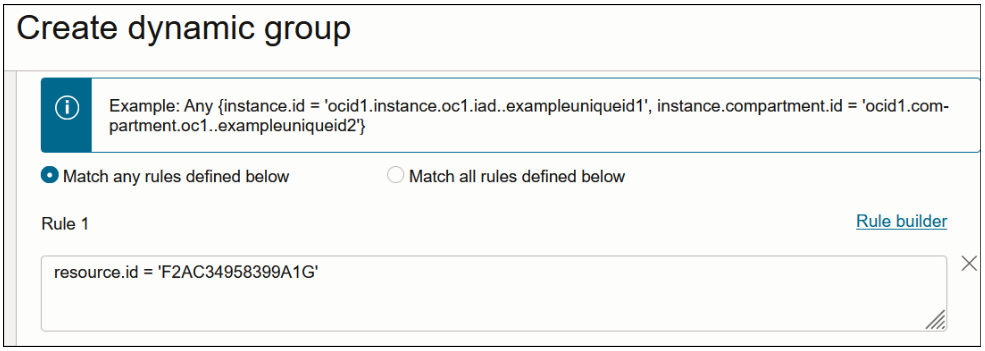
5. Click **Create Dynamic Group**.

6. Enter the following details:

   a. In the **Name** and **Description** fields, enter values. These fields are required.

   b. In the **Matching Rules** section, enter the required rule. The resource ID you specify must match the client ID of the OAuth application of your Oracle Integration instance. Ensure that you enclose the value in single quotes. For example:

```
resource.id = 'client_ID'
```



Create dynamic group

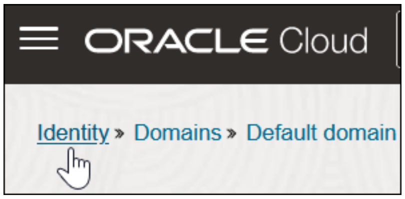
Example: Any {instance.id = 'ocid1.instance.oc1.iad..exampleuniqueid1', instance.compartment.id = 'ocid1.compartment.oc1..exampleuniqueid2'}

● Match any rules defined below ○ Match all rules defined below

Rule 1

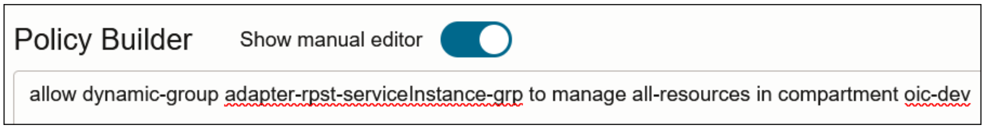
resource.id = 'F2AC34958399A1G'

7. Scroll to the breadcrumbs at the top and click **Identity**.



8. In the left navigation pane, click **Policies**.
9. Click **Create Policy**.
10. Select the compartment in which to create the policy.
11. Enter the following details:
    a. In the **Name** and **Description** fields, enter values. These fields are required.
    b. In the **Policy Builder** section, build the required policy for the dynamic group. For this example, the policy is specified to manage all Oracle Cloud Infrastructure service resources in the compartment in which your Oracle Integration instance is located.

```
allow dynamic-group dynamic_group to manage all-resources in
compartment compartment_name
```



Policy Builder Show manual editor

allow dynamic-group adapter-rpst-serviceInstance-grp to manage all-resources in compartment oic-dev

Where:

– *dynamic_group*: Is the dynamic group name you specified in Step 5.

  - `compartment_name`: Is the compartment in which your Oracle Integration instance is located.

  This enables the Oracle Integration instance associated with the dynamic group to call any Oracle Cloud Infrastructure services in this particular compartment. The RPST token is only valid for resources to which the dynamic group has been granted access using this policy.

JWT Assertions Outbound Use

Perform the following prerequisites to use JWT assertions.

- Manually create a signing key for upload on the Certificates page. See Upload a Certificate to Connect with External Services.
  The service provider typically provides instructions on how to generate the signing keys and the format. For an example, see Required Keys and OCIDs.
- Create the JWT header and JWT payload JSON files. You upload both files on the Connections page when configuring the REST Adapter to support JWT assertions. For example:

JWT Header JSON File Example	JWT Payload JSON File Example

`{`
`  "alg": "RS256",`
`  "kid": "cecsassertion"`
`}`
Where:
– `alg`: The algorithm to use.
– `kid`: A key identifier that is uniquely-generated and associated with the uploaded signing key.
	`{`
	`  "iss": "3485233d3fed4fbcb38338e536d399fc",`
	`  "sub": "3485233d3fed4fbcb38338e536d399fc",`
	`  "aud": "https://identity.oraclecloud.com/",`
	`  "exp": 1672564057"`
	`}`
	Where:
	– JWT issuer (`iss`): A unique identifier for the entity that issued the assertion. This is typically the entity that holds the key material used to sign or integrity-protect the assertion. Examples of issuers are OAuth clients (when assertions are self-issued) and third-party security token services. If the assertion is self-issued, the issuer value is the client identifier (`client_id`). If the assertion was issued by a security token service (STS), the issuer must identify the STS in a manner recognized by the authorization server. The assertion must contain an issuer.
	– JWT subject (`sub`): The subject typically identifies an authorized accessor for which the access token is being requested (that is, the resource owner or an authorized delegate). In some cases, this may be a pseudo anonymous identifier or other value denoting an anonymous user. When the client is acting on behalf of itself, the subject must be the value of the client's `client_id`. The assertion must contain a subject.
	– JWT audience (`aud`): A value that identifies the party or parties to process the assertion. The assertion must contain an audience that identifies the authorization server as the intended audience. The authorization server must reject any assertion that does not contain its own identity as the intended audience (in this case, for the identity domain).
	– Expires at (`exp`): The time at which the assertion expires. While the serialization may differ by assertion format, the time must be expressed in UTC format with no time zone component. The assertion must contain an expires-at entity that limits the window during which the assertion can be used. The authorization server must reject expired assertions (subject to allowable clock skew between systems). The authorization server may reject assertions

JWT Header JSON File Example	JWT Payload JSON File Example
	with an expires-at attribute value that is unreasonably far in the future.

Note:

Carefully review the JWT documentation of your service provider and ensure that you follow all guidelines required by the service provider to correctly create the header and payload files. The content of each file varies from one service provider to another. The REST Adapter supports the different implementations of JWT provided by the following service providers:

- NHS (See Application-restricted RESTful APIs - signed JWT authentication and Step 4: Register your public key)
- FHIR (See Using OAuth 2.0 and Standalone Launch)
- DocuSign (See How to get an access token with JWT Grant)
- Adobe eSign (See JWT (Service Account) Authentication)
- Microsoft (See Microsoft identity platform and OAuth 2.0 On-Behalf-Of flow)
- Okta (See Implement OAuth for Okta with a service app and JWT with private key)

Prerequisites for Identity Propagation Between the REST Adapter and Oracle NetSuite

You must perform the following prerequisites to propagate the user identity from a REST Adapter trigger connection to Oracle NetSuite.

- Complete the prerequisites for the Oracle NetSuite Adapter token-based authentication (TBA) security policy. During this process, you generate values for the token ID and token secret that you specify when creating an Oracle NetSuite Adapter invoke connection. See Prerequisites for the Token-Based Authentication Security Policy in *Using the Oracle NetSuite Adapter with Oracle Integration 3*.
- Create an Oracle NetSuite Adapter invoke connection. During creation, you specify parameter values, such as the token ID and token secret, you created in Oracle NetSuite. See Configure Connection Security in *Using the Oracle NetSuite Adapter with Oracle Integration 3*.
- Create a confidential application in the Oracle Cloud Infrastructure Console. Select and configure the client credentials grant type. The custom JWT claims used to propagate the user identity to Oracle NetSuite work with this grant type. This task requires administrator privileges.
  See Use OAuth 2.0 Grants in Identity Domain Environments in *Using the REST Adapter with Oracle Integration 3*.
- Create custom claims in the Oracle Cloud Infrastructure Identity and Access Management identity domain associated with your Oracle Integration service instance. See Managing Custom Claims. This task requires administrator privileges in your Oracle Cloud Infrastructure Identity and Access Management identity domain. To generate an access token to access these resources, see IAM Identity Domains API.

Create a Connection

Before you can build an integration, you must create the connections to the applications with which you want to share data.

> **Note:**
>
> You can also create a connection in the integration canvas. See Define Inbound Triggers, Outbound Invokes, and Actions.

To create a connection in Oracle Integration:

1. Decide where to start:
   - Work in a project (see why working with projects is preferred).
     a. In the navigation pane, click **Projects**.
     b. Select the project name.
     c. Click **Integrations** 🧩.
     d. In the **Connections** section, click **Add** if no connections currently exist or **+** if connections already exist. The Create connection panel opens.
   - Work outside a project.
     a. In the navigation pane, click **Design**, then **Connections**.
     b. Click **Create**. The Create connection panel opens.
2. Select the adapter to use for this connection. To find the adapter, scroll through the list, or enter a partial or full name in the **Search** field.
3. Enter the information that describes this connection.

Element	Description
Name	Enter a meaningful name to help others find your connection when they begin to create their own integrations.
Identifier	Automatically displays the name in capital letters that you entered in the **Name** field. If you modify the identifier name, don't include blank spaces (for example, `SALES OPPORTUNITY`).

Element	Description
Role	Select the role (direction) in which to use this connection. **Note**: *Only* the roles supported by the adapter you selected are displayed for selection. Some adapters support all role combinations (trigger, invoke, or trigger and invoke). Other adapters support fewer role combinations. When you select a role, only the connection properties and security policies appropriate to that role are displayed on the Connections page. If you select an adapter that supports both invoke and trigger, but select only one of those roles, you'll get an error when you try to drag the adapter into the section you didn't select. For example, assume you configure a connection for the Oracle Service Cloud (RightNow) Adapter as only an **invoke**. Dragging the adapter to a **trigger** section in the integration produces an error.
Keywords	Enter optional keywords (tags). You can search on the connection keywords on the Connections page.
Description	Enter an optional description of the connection.
Share with other projects	**Note**: This field only appears if you are creating a connection in a project. Select to make this connection publicly available in other projects. Connection sharing eliminates the need to create and maintain separate connections in different projects. When you configure an adapter connection in a different project, the **Use a shared connection** field is displayed at the top of the Connections page. If the connection you are configuring matches the same type and role as the publicly available connection, you can select that connection to reference (inherit) its resources. See Add and Share a Connection Across a Project.

4. Click **Create**.

   Your connection is created. You're now ready to configure the connection properties, security policies, and (for some connections) access type.

5. Follow the steps to configure a connection.
   The connection property and connection security values are specific to each adapter. Your connection may also require configuration with an access type such as a private endpoint or an agent group.

6. Test the connection.

Configure Connection Properties for Invoke Connections

Configure connection security to invoke a protected target service with the REST Adapter.

1. Go to the **Properties** section.

2. Specify the following details.

Element	Description
Connection Type	Select the type to use: - **REST API Base URL** - **Open API (1.0/2.0/3.0) URL**

Connection URL	Specify the endpoint URL to use based on your selection in the **Connection Type** field. The connection URL can be both HTTP and HTTPS.
	Note: Do *not* include a trailing forward slash (/) at the end of the REST API Base URL or Open API URL.
	- REST API Base URL
	`https://hostname:port/ic/api/integration/v1/flows/rest/INTEGRATION_NAME/v01`
	- For Open API (1.0/2.0/3.0) URL:
	`https://hostname:port/ic/api/integration/v1/flows/rest/INTEGRATION_NAME/v1/metadata/openapi`

TLS Version (Under **Optional properties**.)	If no value is selected, the default value used for outbound connections is Transport Layer Security (TLS) version 1.3. It's up to your discretion and the end application's requirements to select either TLS version 1.2 or 1.1 as the default. - **TLSv1.1** - **TLSv1.2** TLSv1 is no longer supported. If you previously configured a connection in a version prior to Oracle Integration 3 to use TLSv1.1, either update the connection by not selecting a value for this field or select TLSv1.2. The TLS protocol provides privacy and data integrity between two communicating computer applications. For trigger-only connections, you cannot select a TLS version. Oracle Integration accepts what it receives as long as it's TLSv1.1 or TLSv1.2.
Enable two way SSL for outbound connections (Optional) (Under **Optional properties**.)	If you are configuring the REST Adapter for use with a two-way SSL-enabled server, select **Yes**. .
Identity keystore alias name (Optional) (Under **Optional properties**.)	Enter the key alias name from the keystore file that you specified when importing the identity certificate. The alias name to provide must match the name provided for the private key entry in the JKS file.

Note:

The Metadata Catalog URL, Swagger Definition URL, and RAML Definition URL connection types are no longer available. Developers with a REST API that is described using RAML or the Oracle metadata catalog must take specific actions. See Differences from Prior Versions of Oracle Integration in *What's New for Oracle Integration 3*.

Configure Connection Security

Configure security for your REST Adapter connection by selecting the security policy and specifying the required details.

1. Go to the **Security** section.
2. Select the security policy to use. If you selected the **Invoke** role or the **Trigger and Invoke** role during REST Adapter connection creation, the page is refreshed to display various login credential fields. You must already have created your client application to complete the necessary fields.
   The following security policy restrictions apply when configuring a REST Adapter connection with the trigger and invoke role on the Connections page:
   - If you select Basic Authentication, it can be used as a trigger and an invoke.
   - If you select any other security policy, it can only be used as an invoke. Dragging the connection to the trigger area causes an exception error to be displayed.
   - For existing integrations, the above restrictions do not apply when editing the REST Adapter in the Adapter Endpoint Configuration Wizard.

   > **Note:**
   >
   > The following standard OAuth security policies are implemented to work with providers that are implemented as illustrated in RFC 6749.
   >
   > - OAuth Resource Owner Password Credentials
   > - OAuth Client Credentials
   >
   > In case the standard policy doesn't work, it is recommended that you use the OAuth Custom Two Legged or OAuth Custom Three Legged security policy.

- Configure Security Policies for Trigger Connections
- Configure Security Policies for Invoke Connections

Configure Security Policies for Trigger Connections

Selected Security Policy	Description	Fields
OAuth2.0	• Supports HTTP bearer authentication. • The client should send the OAuth 2.0 bearer token in the HTTP headers. See Authenticate Requests for Invoking Oracle Integration Flows.	No fields are displayed.
Basic Authentication	• Supports HTTP basic authentication. • The client should send the user name/password in the HTTP headers.	No fields are displayed.

Selected Security Policy	Description	Fields
OAuth 2.0 or Basic Authentication	The client can use any of the OAuth 2.0 bearer tokens or the HTTP Basic Authentication header.	No fields are displayed.

Configure Security Policies for Invoke Connections

> **Note:**
>
> OAuth Authorization Code Credentials, OAuth Custom Three Legged Flow, and OAuth Custom Two Legged Flow security types, the connection is only successful after you click the **Provide Consent** button. Configuring all the details alone is not sufficient.

> **Note:**
>
> Testing a REST Adapter connection configured with the HTTP basic authentication security policy and a role connection of **Trigger and Invoke** or **Invoke** does not validate the credentials and simply opens a connection to the provided URL. To validate the endpoint and credentials, the REST Adapter must invoke an API that is idempotent.

Selected Security Policy	Fields
AWS Signature Version 4 **Note**: You can use this security policy with the connectivity agent for scenarios in which you need to invoke AWS APIs hosted in an on-premises environment.	• **Access Key** — Enter the key obtained when you created your Amazon security credentials. • **Secret Key** — Enter the key obtained when you created your Amazon security credentials. • **Confirm Secret Key** — Enter the key a second time. • **AWS Region** — Select the region in which the AWS server is hosted. • **Service Name** — Select the AWS service to which to connect.
Basic Authentication	• **Username** — The name of a user who has access to the destination web service. • **Password** — Enter the password. • **Confirm Password** — Reenter the password.

Selected Security Policy	Fields
OAuth Client Credentials	• **Access Token URI** — The URL from which to obtain the access token. • **Client Id** — The client identifier issued to the client during the registration process. • **Client Secret** — The client secret. • **Confirm Client Secret** — Reenter the client secret. • **Scope** — The scope of the access request. Scopes enable you to specify which type of access you need. Scopes limit access for the OAuth token. They do not grant any additional permission beyond that which the user already possesses. • **Auth Request Media Type** — The format of the data you want to receive. This is an optional parameter that can be kept blank. For example, if you are invoking Twitter APIs, you do not need to select any type. • **Client Authentication** — You can optionally configure OAuth flows with client authentication. This is similar to the Postman user interface feature for configuring client authentication. – **Send client credentials as basic auth header**: Pass the client ID and client secret in the header as basic authentication. – **Send client credentials in body**: Pass the client ID and client secret in the body as form fields.
OAuth Resource Owner Password Credentials	• **Access Token URI** — The URL from which to obtain the access token. • **Client Id** — The client identifier issued to the client during the registration process. • **Client Secret** — The client secret. • **Confirm Client Secret** — Reenter the client secret. • **Scope** — The scope of the access request. Scopes enable you to specify which type of access you need. Scopes limit access for the OAuth token. They do not grant any additional permission beyond that which the user already possesses. • **Auth Request Media Type** — The format of the data you want to receive. • **Username** — The resource owner’s user name. • **Password** — The resource owner’s password. • **Confirm Password** — Reenter the password. • **Client Authentication** — You can optionally configure OAuth flows with client authentication. This is similar to the Postman user interface feature for configuring client authentication. – **Send client credentials as basic auth header**: Pass the client ID and client secret in the header as basic authentication. – **Send client credentials in body**: Pass the client ID and client secret in the body as form fields.

Selected Security Policy	Fields
OAuth Authorization Code Credentials	• **Client Id** — The client identifier issued to the client during the registration process. • **Client Secret** — The client secret. • **Confirm Client Secret** — Reenter the client secret. • **Authorization Code URI** — The URI from which to request the authorization code. • **Access Token URI** — URI to use for the access token. • **Scope** — The scope of the access request. Scopes enable you to specify which type of access you need. Scopes limit access for the OAuth token. They do not grant any additional permission beyond that which the user already possesses. • **Client Authentication** — You can optionally configure OAuth flows with client authentication. This is similar to the Postman user interface feature for configuring client authentication. – **Send client credentials as basic auth header**: Pass the client ID and client secret in the header as basic authentication. – **Send client credentials in body**: Pass the client ID and client secret in the body as form fields.

Selected Security Policy	Fields

OAuth Custom Three Legged Flow
See Configure the REST Adapter to Consume a REST API Protected with OAuth Custom Three Legged Flow Token-Based Authentication to learn more about this security policy.
	- **Authorization Request** — The client application URL to which you are redirected when you provide consent. The authorization server sends a callback to Oracle Integration to obtain an access token for storage. When you create your client application, you must register a redirect URI where the client application is listening.
	`authorization_URI?query_parameters`
	If you don't want to enter values in clear text for this field, do the following:
	– Specify the syntax using an available parameter: `${security_field_1}` through `${security_field_4}` (for example, define scopes as `&scope=${security_field_1}`). See Data Obfuscation Support.
	– Expand **Optional security** and enter the actual values for the parameters you used. To prevent detection, the values you enter are obfuscated. See Security Fields 1 - 4.
	- **Access Token Request** — The access token request to use to fetch the access token. Specify the request using CURL syntax. For example:
	`-X POST method -H headers -d string_data access_token_uri?query_parameters`
	If you don't want to enter values in clear text for this field, do the following:
	– Specify the syntax using an available parameter: `${security_field_1}` through `${security_field_4}` (for example, define the client ID as `&client_id=${security_field_3}`).
	– Expand **Optional security** and enter the actual values for the parameters you used. To prevent detection, the values you enter are obfuscated. See Security Fields 1 - 4.
	- **Refresh Token Request** — The refresh token request to use to fetch the access token. This request refreshes the access token if it expires. Specify the request using CURL syntax. For example
	`-X POST method -H headers -d string_data refresh_token_uri?query_parameters`
	If you don't want to enter values in clear text for this field, do the following:
	– Specify the syntax using an available parameter: `${security_field_1}` through `${security_field_4}` (for example, define client ID as `&client_id=${security_field_2}`).
	– Expand **Optional security** and enter the actual values for the parameters you used. To prevent detection, the values you enter are obfuscated. See Security Fields 1 - 4.

Selected Security Policy	Fields

	• **Sauth_code** — Use regex to identify the authorization code.
	`code`
	• **Saccess_token** — Use a regular expression (regex) to retrieve the access token.
	`access.[tT]oken`
	• **Srefresh_token** — Use regex to retrieve the refresh token.
	`refresh.[tT]oken`
	• **Sexpiry** — Use regex to identify when the access token expires.
	`expires_in`
	• **Stoken_type** — Use regex to identify the access token type.
	`token.?[tT]ype`
	• **access_token_usage** — Specify how to pass the token as multiple headers or multiple query parameters to access a protected resource. You cannot pass a mix of headers and query parameters.
	For headers:
	`-H Authorization: ${token_type} ${access_token} -H validity: 30000 -H signature: ok`
	You can optionally specify quotes for headers:
	`-H 'Authorization: ${token_type} ${access_token}' -H 'validity: 30000' -H 'signature: ok'`
	For query parameters:
	`?token=${access_token}&validity=3000&signature=ok`
	• **Security Fields 1 - 4**
	Enter the actual values for any parameters you used. To prevent detection, the values you enter are obfuscated.
	– **Security Field 1** — Enter the value for the `${security_field_1}` parameter.
	– **Security Field 2** — Enter the value for the `${security_field_2}` parameter.
	– **Security Field 3** — Enter the value for the `${security_field_3}` parameter.
	– **Security Field 4** — Enter the value for the `${security_field_4}` parameter.

Selected Security Policy	Fields

OAuth Custom Two Legged Flow
See Configure the REST Adapter to Consume a REST API Protected with OAuth Custom Two Legged Token-Based Authentication to learn more about this security policy.
	• **Access Token Request** — The access token request to use to fetch the access token. Specify the request using CURL syntax. For example:
	`-X POST method -H headers -d string_data access_token_uri?query_parameters`
	If you don't want to enter values in clear text for this field, do the following:
	– Specify the syntax using an available parameter: `${security_field_1}` through `${security_field_4}` (for example, define client ID as `&client_id=${security_field_1}`). See Data Obfuscation Support.
	– Expand **Optional security** and enter the actual values for the parameters you used. To prevent detection, the values you enter are obfuscated. See Security Fields 1 - 4.
	• **Refresh Token Request** — The refresh token request to use to fetch the access token. This request refreshes the access token if it expires. Specify the request using CURL syntax. For example
	`-X POST method -H headers -d string_data refresh_token_uri?query_parameters`
	If you don't want to enter values in clear text for this field, do the following:
	– Specify the syntax using an available parameter: `${security_field_1}` through `${security_field_4}` (for example, define the client secret as `&client_secret=${security_field_3}`). See Data Obfuscation Support.
	– Expand **Optional security** and enter the actual values for the parameters you used. To prevent detection, the values you enter are obfuscated. See Security Fields 1 - 4.
	• **Saccess_token** — Use regex to identify the access token.
	`access.[tT]oken`
	• **Srefresh_token** — Use regex to identify the refresh token.
	`refresh.[tT]oken`
	• **Sexpiry** — Use regex to identify when the access token expires.
	`expires_in`
	• **Stoken_type** — Use regex to identify the access token type.
	`token.?[tT]ype`
	• **access_token_usage** — Specify how to pass the token as multiple headers or multiple query parameters to access a protected resource. You cannot pass a mix of headers and query parameters.

Selected Security Policy	Fields

	For headers:
	`-H Authorization: ${token_type} ${access_token} -H validity: 30000 -H signature: ok`
	You can optionally specify quotes for headers:
	`-H 'Authorization: ${token_type} ${access_token}' -H 'validity: 30000' -H 'signature: ok'`
	For query parameters:
	`?token=${access_token}&validity=3000&signature=ok`
	• **Security Fields 1 - 4**
	Enter the actual values for any parameters you used. To prevent detection, the values you enter are obfuscated.
	– **Security Field 1** — Enter the value for the `${security_field_1}` parameter.
	– **Security Field 2** — Enter the value for the `${security_field_2}` parameter.
	– **Security Field 3** — Enter the value for the `${security_field_3}` parameter.
	– **Security Field 4** — Enter the value for the `${security_field_4}` parameter.

API Key Based Authentication
See Configure the REST Adapter to Consume a REST API Protected with the API Key to learn more about this security policy.
	• **API Key** — Specify the generated API key used to identify the client making the request.
	• **Confirm API Key** — Reenter the API key.
	• **API Key Usage** — Specify the URI syntax for how to pass the API key to access a protected resource.
	To pass the API key as a query parameter at runtime to access the protected resource:
	`?key=${api-key}`
	To pass the API key as a header at runtime to access the protected resource.
	`-H Authorization: Bearer ${api_key}`
	For example:
	`-H Authorization: Bearer AASDFADADX`

Selected Security Policy	Fields
OAuth 1.0 One Legged Authentication	- **Consumer Key** — Specify the key that identifies the client making the request. - **Consumer Secret** — Specify the consumer secret that authorizes the client making the request. - **Confirm Consumer Secret** — Specify the secret a second time. - **Token** — Specify the token that accesses protected resource. - **Token Secret** — Specify the token secret that generates the signature for the request. - **Confirm Token Secret** — Specify the secret a second time. - **Realm** — Specify the realm that identifies the account. **Note**: The **HMAC-SHA256** signature encryption algorithm is supported by default and cannot be changed. **HMAC-SHA1** is not supported in Oracle Integration 3.
OCI Signature Version 1	Specify the values you created when satisfying the prerequisites for using this security policy. See Prerequisites for Creating a Connection. - **Tenancy OCID** — Specify the value you copied from the Oracle Cloud Infrastructure Console. - **User OCID** — Specify the value you copied from the Oracle Cloud Infrastructure Console. - **Private Key** — Click **Upload** to select the key you created. Ensure that the key is in RSA (PKCS1) format. If you need to convert to this format, see Convert a Private Key from PKCS8 to RSA (PKCS1) Format for the OCI Signature Version 1 Security Policy. - **Finger Print** — Enter the finger print that was generated when you created the key in the Oracle Cloud Infrastructure Console. - **Pass Phrase** — Enter the pass phrase you created when creating the key. - **Confirm Pass Phrase** — Enter the pass phrase a second time.
OAuth Client Credentials using JWT Client Assertion **Note**: This policy is typically used to invoke application-driven APIs.	- **Access token URI** — Enter the URL to which to send a request to obtain the access token. For example: `https://accounts.google.com/o/oauth2/token` - **JWT headers in JSON format** — Upload the JWT header file in JSON format. - **JWT payload in JSON format** — Upload the JWT payload file in JSON format. - **JWT private key alias** — Enter the JWT private key alias. This is the same alias you specified when uploading the signing key certificate on the Certificates page. - **Scope** — (Optional) Enter the scopes. - **Access token request** — (Optional) Enter the request to obtain the access token. The format you specify can vary by service provider. See Variations of JWT Usage by Service Providers.

Selected Security Policy	Fields
OAuth using JWT User Assertion **Note**: • This policy is typically used on behalf of a user. • This policy supports the propagation of a user identity between services. See Propagate OAuth User Identity Between Services.	• **Access token URI** — Enter the URL to which to send a request to obtain the access token. For example: `https://accounts.google.com/o/oauth2/token` • **JWT headers in JSON format** — Upload the JWT header file in JSON format. • **JWT payload in JSON format** — Upload the JWT payload file in JSON format. • **JWT private key alias** — Enter the JWT private key alias. This is the same alias you specified when uploading the signing key certificate on the Certificates page. • **Scope** — (Optional) Enter the scopes. • **Access token request** — (Optional) Enter the request to obtain the access token. The format you specify can vary by service provider. See Variations of JWT Usage by Service Providers.
OCI Service Invocation	After selecting this security policy, you are not prompted to specify any values. Configuration is automatic. However, you must perform all prerequisites for configuration to succeed. See RPST and OCI Service Invocation Security Policy Use.
No Security Policy	If you select this security policy, no additional fields are displayed.

Variations of JWT Usage by Service Providers

Service providers implement JWT assertions in different ways, including how to specify the scope value and an access token request value in the **Scope** and **Access token request** fields when configuring the OAuth Client Credentials using JWT Client Assertion or OAuth using JWT User Assertion security policy on the Connections page.

Service Provider	Requires Provide Consent ?	Scope and Access token request Fields on Connections Page	Reference Documentation

Okta	No	`curl --location --request POST 'https://${yourOktaDomain}/oauth2/v1/token' \`
		`   --header 'Accept: application/json' \`
		`   --header 'Content-Type: application/x-www-form-urlencoded' \`
		`   --data-urlencode '`**`grant_type`**`=client_credentials' \`
		`   --data-urlencode '`**`scope`**`=okta.users.read' \`
		`   --data-urlencode '`**`client_assertion_type`**`=urn:ietf:params:oauth:client-assertion-type:jwt-bearer' \`
		`   --data-urlencode '`**`client_assertion`**`=eyJhbGciOiJSU....tHQ6ggOnrG-ZFRSkZc8Pw'`
			Implement OAuth for Okta with a service app

Service Provider	Requires Provide Consent ?	Scope and Access token request Fields on Connections Page	Reference Documentation

Okta	Yes	POST /token HTTP/1.1
		Host: server.example.com
		Content-Type: application/x-www-form-urlencoded
		grant_type=authorization_code&
		  code=<id_token>&
		  client_id=<client_id>
		  client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer&
		  client_assertion=<client_assertion>
			JWT with private key

NHS	no	curl -x post -h "content-type:application/x-www-form-urlencoded" --data \ "**grant_type**=client_credentials\ &**client_assertion_type**=urn:ietf:params:oauth:client-assertion-type:jwt-bearer\ &**client_assertion**=<your-signed-jwt>" \ https://api.service.nhs.uk/oauth2/token	Application-restricted RESTful APIs - signed JWT authentication
NHS	Yes	curl --location --request POST 'https://api.service.nhs.uk/oauth2/token'\ --header 'Content-Type: application/x-www-form-urlencoded'\ --data-urlencode '**grant_type**=urn:ietf:params:oauth:grant-type:token-exchange'\ --data-urlencode '**subject_token_type**=urn:ietf:params:oauth:token-type:id_token'\ --data-urlencode '**client_assertion_type**=urn:ietf:params:oauth:client-assertion-type:jwt-bearer'\ --data-urlencode '**subject_token**={NHS CIS2 ID token} \ --data-urlencode '**client_assertion**={jwt token}	Step 4: Register your public key User-restricted RESTful APIs - NHS login separate authentication and authorization
FHIR	No	POST https://fhir.epic.com/interconnect-fhir-oauth/oauth2/token HTTP/1.1 Content-Type: application/x-www-form-urlencoded **grant_type**=client_credentials&**client_assertion_type**=urn:ietf:params:oauth:client-assertion-type:jwt-bearer&**client_assertion**=<client_assertion>	Using OAuth 2.0

Service Provider	Requires Provide Consent ?	Scope and Access token request Fields on Connections Page	Reference Documentation
FHIR	Yes	POST https://fhir.epic.com/interconnect-fhir-oauth/oauth2/token HTTP/1.1 Content-Type: application/x-www-form-urlencoded grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=[assertion]&client_id=[client_id]	Standalone Launch
Microsoft	No	POST /{tenant}/oauth2/v2.0/token HTTP/1.1 Host: login.microsoftonline.com Content-Type: application/x-www-form-urlencoded **scope**=https://graph.microsoft.com/.default &**client_id**=97e0a5b7-d745-40b6-94fe-5f77d35c6e05 &**client_assertion_type**=urn:ietf:params:oauth:client-assertion-type:jwt-bearer &**client_assertion**=<client_assertion> &**grant_type**=client_credentials	Microsoft identity platform and the OAuth 2.0 client credentials flow
Microsoft	Yes	POST /oauth2/v2.0/token HTTP/1.1 Host: login.microsoftonline.com/<tenant> Content-Type: application/x-www-form-urlencoded **grant_type**=urn:ietf:params:oauth:grant-type:jwt-bearer& **client_id**=<client_id>& **client_assertion_type**=urn:ietf:params:oauth:client-assertion-type:jwt-bearer&**client_assertion**=<client_assertion> &assertion=<assertion>&**requested_token_use**=on_behalf_of &**scope**=https://graph.microsoft.com/user.read+offline_access	Microsoft identity platform and OAuth 2.0 On-Behalf-Of flow
DocuSign	Yes	curl --data "**grant_type**=urn:ietf:params:oauth:grant-type:jwt-bearer& **assertion**=YOUR_JSON_WEB_TOKEN" --request POST https://account-d.docusign.com/oauth/token	How to get an access token with JWT Grant

Service Provider	Requires Provide Consent?	Scope and Access token request Fields on Connections Page	Reference Documentation
Adobe	No	POST https://ims-na1.adobelogin.com/ims/exchange/jwt **client_id**={api_key_value}&**client_secret**={client_secret_value}&**jwt_token**={base64_encoded_JWT}	JWT (Service Account) Authentication
Oracle Cloud Infrastructure Identity and Access Management (identity domains)	No	POST <hostname>/oauth2/v1/token grant_type=client_credentials&client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer&client_assertion=<client_assertion>&scope=<scope>	Client/ User JWT Assertion.
Oracle Cloud Infrastructure Identity and Access Management (identity domains)	No	**grant_type**=urn:ietf:params:oauth:grant-type:jwt-bearer&**assertion**=<user_assertion>&**scope**=<scope>&**client_assertion_type**=urn:ietf:params:oauth:Aclient-assertion-type:jwt-bearer& **client_assertion**=<client_assertion>	Client/ User JWT Assertion.

Configure the Endpoint Access Type

Configure access to your endpoint. Depending on the capabilities of the adapter you are configuring, options may appear to configure access to the public internet, to a private endpoint, or to an on-premises service hosted behind a fire wall.

- Select the Endpoint Access Type
- Ensure Private Endpoint Configuration is Successful

Select the Endpoint Access Type

1. Go to the **Access type** section.
2. Select the option for accessing your endpoint.

Option	This Option Appears If Your Adapter Supports ...
Public gateway	Connections to endpoints using the public internet.
Private endpoint	Connections to endpoints using a private virtual cloud network (VCN). **Note**: To connect to private endpoints, you must complete prerequisite tasks in the Oracle Cloud Infrastructure Console. Failure to do so results in errors when testing the connection. See Connect to Private Resources in *Provisioning and Administering Oracle Integration 3* and Troubleshoot Private Endpoints in *Using Integrations in Oracle Integration 3*.
Connectivity agent	Connections to on-premises endpoints through the connectivity agent. a. Click **Associate agent group**. The Associate agent group panel appears. b. Select the agent group, and click **Use**. To configure an agent group, you must download and install the on-premises connectivity agent. See Download and Run the Connectivity Agent Installer and About Creating Hybrid Integrations Using Oracle Integration in *Using Integrations in Oracle Integration 3*.

Ensure Private Endpoint Configuration is Successful

- To connect to private endpoints, you must complete prerequisite tasks in the Oracle Cloud Infrastructure Console. Failure to do so results in errors when testing the connection. See Connect to Private Resources in *Provisioning and Administering Oracle Integration 3*.
- When configuring an adapter on the Connections page to connect to endpoints using a private network, specify the fully-qualified domain name (FQDN) and *not* the IP address. If you enter an IP address, validation fails when you click **Test**.
- IPSec tunneling and FastConnect are not supported for use with private endpoints.

Test the Connection

Test your connection to ensure that it's configured successfully.

1. In the page title bar, click **Test**. What happens next depends on whether your adapter connection uses a Web Services Description Language (WSDL) file. Only some adapter connections use WSDLs.

If Your Connection...	Then...
Doesn't use a WSDL	The test starts automatically and validates the inputs you provided for the connection.

If Your Connection...	Then...
Uses a WSDL	A dialog prompts you to select the type of connection testing to perform: • **Validate and Test**: Performs a full validation of the WSDL, including processing of the imported schemas and WSDLs. Complete validation can take several minutes depending on the number of imported schemas and WSDLs. No requests are sent to the operations exposed in the WSDL. • **Test**: Connects to the WSDL URL and performs a syntax check on the WSDL. No requests are sent to the operations exposed in the WSDL.

2. Wait for a message about the results of the connection test.
   - If the test was successful, then the connection is configured properly.
   - If the test failed, then edit the configuration details you entered. Check for typos and verify URLs and credentials. Continue to test until the connection is successful.
3. When complete, click **Save**.

Upload a Certificate to Connect with External Services

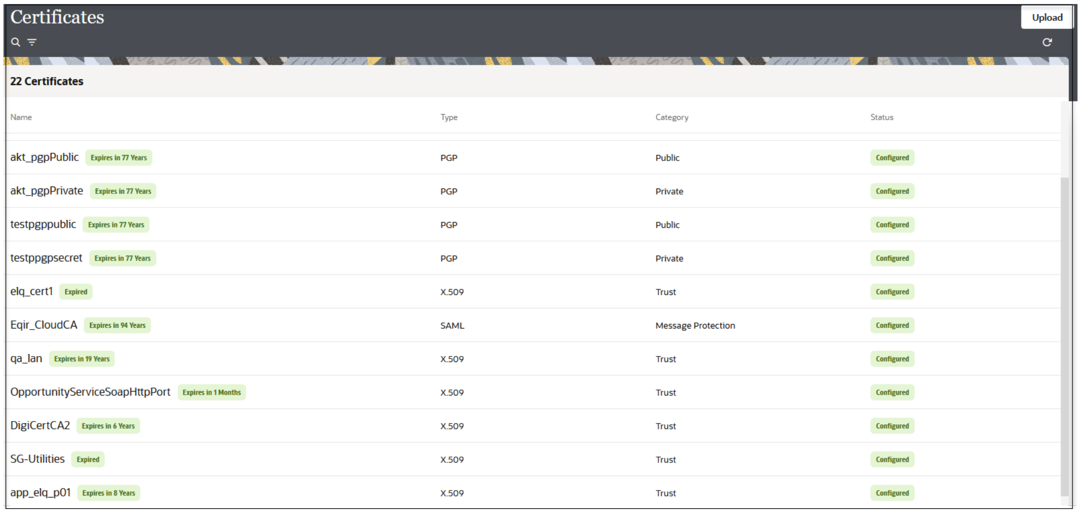
Certificates allow Oracle Integration to connect with external services. If the external service/ endpoint needs a specific certificate, request the certificate and then import it into Oracle Integration.

If you make an SSL connection in which the root certificate does not exist in Oracle Integration, an exception error is thrown. In that case, you must upload the appropriate certificate. A certificate enables Oracle Integration to connect with external services. If the external endpoint requires a specific certificate, request the certificate and then upload it into Oracle Integration.

For the REST Adapter, note the following certificate details for one-way and two-way SSL:

- One-way SSL: Oracle Integration only needs the public certificates of the HTTPS URL. The certificate must be in `.pem` or `.crt` format and uploaded as a trust certificate.
- Two-way SSL: You must add the public certificate of the HTTPS URL. After that, you must create a JKS certificate with a private key file. You must upload the JKS certificate as an identity certificate.
  Once you upload the identity certificate, you must use an alias on the Connections page.

1. Sign in to Oracle Integration.
2. In the navigation pane, click **Settings**, then **Certificates**.
   All certificates currently uploaded to the trust store are displayed on the Certificates page.
3. Click **Filter** to filter by name, certificate expiration date, status, type, category, and installation method (user-installed or system-installed). Certificates installed by the system cannot be deleted.



22 Certificates

Name	Type	Category	Status
akt_pgpPublic Expires in 77 Years	PGP	Public	Configured
akt_pgpPrivate Expires in 77 Years	PGP	Private	Configured
testpgppublic Expires in 77 Years	PGP	Public	Configured
testppgpsecret Expires in 77 Years	PGP	Private	Configured
elq_cert1 Expired	X.509	Trust	Configured
Eqir_CloudCA Expires in 94 Years	SAML	Message Protection	Configured
qa_lan Expires in 19 Years	X.509	Trust	Configured
OpportunityServiceSoapHttpPort Expires in 1 Months	X.509	Trust	Configured
DigiCertCA2 Expires in 6 Years	X.509	Trust	Configured
SG-Utilities Expired	X.509	Trust	Configured
app_elq_p01 Expires in 8 Years	X.509	Trust	Configured

4. Click **Upload** at the top of the page.
   The Upload certificate panel is displayed.
5. Enter an alias name and optional description.
6. In the **Type** field, select the certificate type. Each certificate type enables Oracle Integration to connect with external services.
   - Digital Signature
   - X.509 (SSL transport)
   - SAML (Authentication & Authorization)
   - PGP (Encryption & Decryption)
   - Signing key

Digital Signature

The digital signature security type is typically used with adapters created with the Rapid Adapter Builder. See Learn About the Rapid Adapter Builder in Oracle Integration in *Using the Rapid Adapter Builder with Oracle Integration 3*.

1. Click **Browse** to select the digital certificate. The certificate must be an X509Certificate. This certificate provides inbound RSA signature validation. See RSA Signature Validation in *Using the Rapid Adapter Builder with Oracle Integration 3*.
2. Click **Upload**.

X.509 (SSL transport)

1. Select a certificate category.
   a. **Trust**: Use this option to upload a trust certificate.
      i. Click **Browse**, then select the trust file (for example, `.cer` or `.crt`) to upload.
   b. **Identity**: Use this option to upload a certificate for two-way SSL communication.
      i. Click **Browse**, then select the keystore file (`.jks`) to upload.
      ii. Enter the comma-separated list of passwords corresponding to key aliases.

> **Note:**
>
> When an identity certificate file (`.jks`) contains more than one private key, all the private keys must have the same password. If the private keys are protected with different passwords, the private keys cannot be extracted from the keystore.

    iii. Enter the password of the keystore being imported.

  c. Click **Upload**.

SAML (Authentication & Authorization)

1. Note that **Message Protection** is automatically selected as the only available certificate category and cannot be deselected. Use this option to upload a keystore certificate with SAML token support. Create, read, update, and delete (CRUD) operations are supported with this type of certificate.
2. Click **Browse**, then select the certificate file (`.cer` or `.crt`) to upload.
3. Click **Upload**.

PGP (Encryption & Decryption)

1. Select a certificate category. Pretty Good Privacy (PGP) provides cryptographic privacy and authentication for communication. PGP is used for signing, encrypting, and decrypting files. You can select the private key to use for encryption or decryption when configuring the stage file action.
   a. **Private**: Uses a private key of the target location to decrypt the file.
      i. Click **Browse**, then select the PGP file to upload.
      ii. Enter the PGP private key password.
   b. **Public**: Uses a public key of the target location to encrypt the file.
      i. Click **Browse**, then select the PGP file to upload.
      ii. In the **ASCII-Armor Encryption Format** field, select **Yes** or **No**.
         - **Yes** shows the format of the encrypted message in ASCII armor. ASCII armor is a binary-to-textual encoding converter. ASCII armor formats encrypted messaging in ASCII. This enables messages to be sent in a standard messaging format. This selection impacts the visibility of message content.
         - **No** causes the message to be sent in binary format.
      iii. From the **Cipher Algorithm** list, select the algorithm to use. Symmetric-key algorithms for cryptography use the same cryptographic keys for both encryption of plain text and decryption of cipher text. The following supported cipher algorithms are FIPS-compliant:
         - AES128
         - AES192
         - AES256
         - TDES
   c. Click **Upload**.

Signing key

A signing key is a secret key used to establish trust between applications. Signing keys are used to sign ID tokens, access tokens, SAML assertions, and more. Using a private signing key, the token is digitally signed and the server verifies the authenticity of the token by using a public signing key. You must upload a signing key to use the OAuth Client Credentials using JWT Client Assertion and OAuth using JWT User Assertion security policies in REST Adapter invoke connections. Only PKCS1- and PKCS8-formatted files are supported.

1. Select **Public** or **Private**.
2. Click **Browse** to upload a key file.
   If you selected **Private**, and the private key is encrypted, a field for entering the private signing key password is displayed after key upload is complete.
3. Enter the private signing key password. If the private signing key is not encrypted, you are not required to enter a password.
4. Click **Upload**.

4 Add the REST Adapter Connection to an Integration

When you drag the REST Adapter into the trigger or invoke area of an integration, the Adapter Endpoint Configuration Wizard appears. This wizard guides you through the configuration of the REST Adapter endpoint properties.

These topics describe the wizard pages that guide you through configuration of the REST Adapter as a trigger or invoke in an integration.

> **Note:**
>
> - XML documents passed to a REST endpoint that support the application/XML content type must comply with the XML schema specified during trigger (inbound) REST Adapter configuration. When the REST Adapter invokes a target endpoint, the application/XML response must comply with the XML schema specified during invoke (outbound) REST Adapter response configuration.
> - If the following integrations are imported from one environment to another (having different host names), then editing the local (child) integration or editing the REST Adapter in the Adapter Endpoint Configuration Wizard leads to major changes in the mapper that may require remapping.
>   - Integrations in which a co-located (child) integration is invoked from a parent integration. See Invoke a Co-located Integration from a Parent Integration.
>   - Integrations with a REST adapter using a Swagger-based connection.

Topics:

- Add the REST Adapter as a Trigger Connection
- Add the REST Adapter as an Invoke Connection

Add the REST Adapter as a Trigger Connection

When you drag the REST Adapter into the integration canvas as a trigger connection, the Adapter Endpoint Configuration Wizard is invoked. Based on your selections in the wizard, the following pages can be displayed.

Topics

- REST Adapter Trigger Basic Information Page
- REST Adapter Trigger Resource Configuration Page
- REST Adapter Trigger Configure Custom Claims Page
- REST Adapter Trigger Request Parameters Page
- REST Adapter Trigger Request Page

- REST Adapter Trigger Request Header Page
- REST Adapter Trigger CORS Configuration Page
- REST Adapter Trigger Response Page
- REST Adapter Trigger Response Header Page
- REST Adapter Trigger Operations Page
- REST Adapter Trigger Operation Selection Page
- Summary Page

REST Adapter Trigger Basic Information Page

Enter the REST Adapter user name and description. You can also select to configure multiple resources or verbs.

Element	Description
What do you want to call your endpoint?	Provide a meaningful name so that others can understand the connection. For example, if you are creating a source Oracle REST connection, you may want to name it `ExposeFlowAsRESTResource`. You can include English alphabetic characters, numbers, underscores, and dashes in the name. You cannot include the following: • Blank spaces (for example, `My REST Connection`) • Special characters (for example, `#;83& or res(t)4`) • Multibyte characters
What does this endpoint do?	Enter an optional description of the endpoint's responsibilities (for example,`This inbound endpoint exposes this integration flow as a REST resource`).
Select to configure multiple resources or verbs (maximum 11)	Select to configure multiple operation entry points with different resource URIs and HTTP actions/verbs, as necessary. Each operation represents a different pick action branch in a single integration. The maximum number of operations (branches) you can create in one integration is eleven. This feature eliminates the need to create multiple integrations (each with a separate resource URI and verb) to perform different operations.

REST Adapter Trigger Resource Configuration Page

Enter the REST Adapter operation name, relative resource URI, and endpoint action. You can also select to add query and template parameters or configure a request and/or response for the endpoint. If integrating with Oracle NetSuite, you can select to configure identity propagation using JWT User Assertion.

Element	Description
Provide an operation name	Enter an operation name.
What does this operation do?	Enter an optional description of the operation's responsibilities.

Element	Description

What is the endpoint's relative resource URI?	Specify the relative path associated with the resource. The path can contain template parameters specified with curly braces (for example, `{order-id}`). A resource is any source of specific information that can be addressed. The resource path follows a fixed, prefixed URL appended with the specified relative path. By default, the URL is prefixed with the following path:
	`https://instance_URL/ic/api/integration/v1/flows/rest/INTEGRATION_NAME/VERSION`
	For example, if the integration name is `ExposeFlowAsRESTResource`, the URL becomes:
	`https://instance_URL/ic/api/integration/v1/flows/rest/EXPOSEFLOWASRESTRESOURCE`
	You can override the URL, except for the fixed part at the beginning:
	`instance_URL/ic`

What action do you want to perform on the endpoint?	Select a single HTTP action (method) for the endpoint to perform: • **GET**: Retrieves (reads) information (for example, makes queries). If you select this option, you cannot configure a request payload for this endpoint. • **PUT**: Updates information. • **POST**: Creates information. • **DELETE**: Deletes information. If you select this option, you cannot configure a request payload for this endpoint. **PATCH**: Partially updates existing resources (for example, when you only need to update one attribute of the resource). **Note**: The **PATCH** verb does not work with a non-SSL REST service.

Element	Description
Select any options that you want to configure	Select the options that you want to configure: • **Add and review parameters for this endpoint**: Click to specify the query parameters and view the template request parameters created as part of the resource URI for this endpoint. If you select this option and click **Next**, the Request Parameters page is displayed. • **Configure a request payload for this endpoint**: Click to configure the request payload for this endpoint, including specifying the schema location and payload type with which you want the endpoint to reply. You can also select this option if you want to include an attachment with the inbound request. If you select this option and click **Next**, the Request page is displayed. • **Configure this endpoint to receive the response**: Click to configure the response payload for this endpoint, including specifying the schema location and payload type that you want the endpoint to receive. If you select this option and click **Next**, the Response page is displayed.
Configure Request Headers?	Select the type of request header to configure: • **Standard**: Select to configure standard HTTP headers for the request message. • **Custom**: Select to configure custom HTTP headers for the request message.
Configure Response Headers?	Select the type of response header to configure: • **Standard**: Select to configure standard HTTP headers for the response message. • **Custom**: Select to configure custom HTTP headers for the response message.
Configure CORS (Cross Origin Resource Sharing) (available only in the trigger (inbound) direction)	Select to configure CORS parameters for a trigger. CORS enables restricted resources (for example,. custom HTTP headers that introduce cross-site Java scripting security issues) on a web page to be requested from another domain outside of the domain from which the resource originated.
Configure Custom JWT Claims (available only in the trigger (inbound) direction)	Select to propagate the user identity to Oracle NetSuite with custom JWT claims. **Note**: This selection is only applicable to identity propagation with Oracle NetSuite.

REST Adapter Trigger Configure Custom Claims Page

Add the custom JWT claims to use for client ID and client secret. The custom JWT claims enable you to propagate user identity to Oracle NetSuite. This page only appears if you selected **Configure Custom JWT Claims** on the Resource Configuration page.

Element	Description
Add	Click **Add** to specify the custom JWT claims for client ID and client secret. The claims you add must also be defined in Oracle Cloud Infrastructure Identity and Access Management. Once you complete the Adapter Endpoint Configuration Wizard, the names you entered appear in the **Source** section of the mapper under **Security Properties**, then **Custom Claims**.
Edit	Double-click the row to edit an entry.
Remove	Select the row, then click **Remove**.

A use case is provided of this feature. See Propagate User Identity from the REST Adapter to Oracle NetSuite with Custom JWT Claims.

REST Adapter Trigger Request Parameters Page

Enter the REST Adapter request parameters for this endpoint.

Element	Description
Resource URI	Displays the endpoint relative resource URI entered on the Basic Info page.
Specify Query Parameters	Specify query parameters for the REST endpoint. Click the **Add** icon to display a row for entering the parameter name and selecting its data type. For example, specify `state` and select a data type of **string**. Click the **Delete** icon to delete a selected row.

Element	Description
Template Parameters	Displays the template parameters in the relative resource URI. Template parameters are based on details you specified on the Basic Info page and cannot be edited. Template parameters must be defined as part of a path with curly braces around them. For example, the URL `default/customers/{cust-id}/{ship-id` has `cust-id` and `ship-id` template parameters. You can change the data type for the parameters. **Note**: • Any query and template parameters added or configured are available for mapping in the mapper and in the actions in integrations. • Query and template parameter values added in the URL specified on the Connection page do not appear in the mapper. Instead, the template and query parameters must be configured in the Adapter Endpoint Configuration Wizard for those parameters to appear in the mapper.

REST Adapter Trigger Request Page

Enter the REST Adapter request payload details for the endpoint.

Element	Description
Select the multipart attachment processing options	Configure the following options based on whether the request is inbound or outbound. For inbound (trigger) requests, select the multipart attachment type to include. This option is only available if you selected the POST action on the Basic Info page. • **Request is multipart with payload**: Select to send multipart attachments as part of the request along with JSON or XML content as the payload request. • **Multipart request is of type multipart/form-data with HTML form payload**: Select for the REST endpoint to accept to configure an HTML form. You must first select the **Request is multipart with payload** option before you can select this option. This selection assumes that the media type is multipart/form-data.

Element	Description
Select the request payload format	**Note**: - Ensure that the sample JSON or the uploaded XML schema is representative of the actual runtime messages exchanged with the endpoint. A mismatch in the structure or type of runtime messages can result in errors. - If you upload a schema file without a target namespace, a surrogate namespace is added to the schema file that all messages then use: `http://xmlns.oracle.com/cloud/adapter/nxsd/surrogate` Select the request payload format to use. The request payload body must be defined by the XSD element that defines the structure of this representation. - **XML Schema** - **JSON Sample**: Select this option to use Swagger and RAML files. JSON sample files of up to 100 KB in size are supported. Empty arrays in JSON sample files are not supported. For information, see Empty Arrays Are Not Supported in Sample JSON Files. You may need to process large JSON sample files with special characters before using the Adapter Endpoint Configuration Wizard. See Large Sample JSON File Processing with Special Characters. - **XML Sample (Single or No Namespace)**: Select this option to use an XML document to generate the schema. - **Binary**: Use with payloads that are unstructured and inline — for example, `application/octet-stream`. It preserves the file contents, but requires the receiver to determine file type, for example, from the filename extension. The Internet media type for an arbitrary byte stream is `application/octet-stream`.
Schema Location	Specify the schema file in either of the following ways: - Click **Browse** to select the request schema file to use. - Click **<<inline>>** to copy and paste the JSON payload or URL into a text field. Click **OK** when complete.
Element	Select the element that defines the payload structure. This field is not displayed until you import the request payload file. Once you browse for and select the schema or JSON sample file, the schema is displayed automatically. It also displays a combination box that selects the root element by default.

Element	Description
What is the media-type of the Request Body? (Content-Type Header)	- **XML**: Displays the payload in XML format. - **XML (text)**: Displays the payload in XML text format. - **JSON**: Displays the payload in JavaScript Object Notation (JSON) format. - **URL-encoded**: Displays the payload in URL-encoded format. - **Other Media Type**: Select to display the payload in another format (for example, `application/oracle.cloud+json`). You can only specify the media types that end with `+json` or `+xml`. The following media types are supported implicitly and cannot be configured. At runtime, the request media type is in the form of an `http Content-Type` header. The expected response media type is specified through an `Accept` header. Any service can be accessed through either of these media types. – Application/XML – Application/JSON Select the multipart attachment type for the endpoint to receive. This field is displayed if you selected the **Request is multipart with payload** option in the **Select the multipart attachment processing options** field. - **multipart/mixed**: Send an XML or JSON payload type with an attachment. For example, send a PDF document for review as a link in an email. - **multipart/form-data**: Send an XML or JSON payload type with an attachment. For example, you create an HTML form to upload and send an image. In the HTML form, the `method` is defined as `post` and the `enctype` (encoding type) is defined as `multipart/form-data`. You can also send the attachment alone without a payload when using this attachment type.

REST Adapter Trigger Request Header Page

Enter the REST Adapter request header properties for this endpoint.

> **Note:**
>
> If you specify a custom header name that is the same as a standard header name, an error occurs. Ensure that you specify unique names for your custom headers.

Specify the standard HTTP request headers to use.

Element	Description
Add Standard Request Headers	Select the standard HTTP request header to use from the default dropdown list. • Click the **Add** icon to add an additional row, then select the standard HTTP request header to use from the dropdown list. Standard headers available for selection include, but are not limited to, the following: – **Accept**: When sent by a client, the **Accept** header is published to an integration. This header describes the preferred format in which the client wants to accept the request. This allows for the propagation of the header sent by the client application to Oracle Integration. – **Connection**: If a request is sent with the connection header set, this value is propagated to the integration. – **Content Length**: The length of the content is propagated to Oracle Integration, regardless of content length. Post translation, this value may not match the actual content. – **Content-Type**: This allows for the propagation of the header sent by the client application to Oracle Integration. – **Origin**: This header sent as part of the request to a REST Adapter trigger is propagated to the integration. • Click the **Delete** icon to delete the row of a selected standard HTTP request header. **Note:**
HTTP Header Name	Perform the following tasks: • From the list, select the header to use.

Specify the custom HTTP request headers to use.

Element	Description
Add Custom Request Headers	Perform the following custom request header tasks: • Click the **Add** icon to add custom HTTP request headers and optional descriptions. • Click the **Delete** icon to delete the selected custom HTTP request headers.
Custom Header Name	Enter the custom header name.
Custom Header Description	Enter an optional description.

REST Adapter Trigger CORS Configuration Page

Enter the REST Adapter CORS configuration properties for this endpoint.

Element	Description
Allowed Origins	Specify the allowable domains from which to make CORS requests. Requests coming from these domains are accepted. Enter an asterisk (`*`) for all domains to make the requests. Enter comma-separated values for specific domains to make the requests (for example, `http://localhost:8080` , `https://myhost.example.com:7002`).
Allowed Methods	The allowed method displayed is based on your selection in the **What action does the endpoint perform?** list on the Basic Info page. Requests are only accepted from the allowable domains that perform the allowable actions (methods). You cannot configure the method name listed in the CORS configuration.

REST Adapter Trigger Response Page

Enter the REST Adapter response payload details for the endpoint.

Element	Description
Select the multipart attachment processing options	Configure the following options based on whether the request is inbound or outbound. For inbound (trigger) responses, select the multipart attachment type to include. • **Response is multipart with payload**: Select to receive the response from the payload. • **Multipart response is of type multipart/form-data with HTML form payload**: Select for the REST endpoint to accept to configure an HTML form. You must first select the **Response is multipart with payload** option before you can select this option. This selection assumes that the media type is multipart/form-data.

Element	Description
Select the response payload format	**Note**: • Ensure that the sample JSON or the uploaded XML schema is representative of the actual runtime messages exchanged with the endpoint. A mismatch in the structure or type of runtime messages can result in errors. • If you upload a schema file without a target namespace, a surrogate namespace is added to the schema file that all messages then use: `http://xmlns.oracle.com/cloud/adapter/nxsd/surrogate` Select the response payload format to use. The response payload body must be defined by the XSD element that defines the structure of this representation. • **XML Schema** • **JSON Sample**: Select this option to use Swagger and RAML files. JSON sample files of up to 100 KB in size are supported. Empty arrays in JSON sample files are not supported. For information, see Empty Arrays Are Not Supported in Sample JSON Files. You may need to process large JSON sample files with special characters before using the Adapter Endpoint Configuration Wizard. See Large Sample JSON File Processing with Special Characters. • **XML Sample (Single or No Namespace)**: Select this option to use an XML document to generate the schema. • **Binary**: Use with payloads that are unstructured and inline — for example, `application/octet-stream`. It preserves the file contents, but requires the receiver to determine the file type, for example, from the filename extension. The Internet media type for an arbitrary byte stream is `application/octet-stream`.
Schema Location	Specify the schema file in either of the following ways: • Click **Browse** to select the response schema file to use. • Click **<<inline>>** to copy and paste the JSON payload or URL into a text field. Click **OK** when complete.
Element	Select the element that defines the payload structure. This field is not displayed until you import the response payload file. Once you browse for and select the schema file, it displays a combination box that selects the root element by default.

Element	Description
What is the media-type of Response Body (Accept Header)	Select the payload type with which you want the endpoint to reply. • **XML**: Displays the payload in XML format. • **XML (text)**: Displays the payload in XML text. • **JSON**: Displays the payload in JavaScript Object Notation (JSON) format. • **Other Media Type**: Select to display the payload in another format (for example, `application/oracle.cloud+json`). You can only specify media types that end with `+json` or `+xml`. The following media types are supported implicitly and cannot be configured. At runtime, the request media type is in the form of an `http Content-Type` header. The expected response media type is specified through an `Accept` header. Any service can be accessed through either of these media types. – Application/XML – Application/JSON Select the multipart attachment type for the endpoint to receive. This field is displayed if you selected the **Response is multipart with payload** option in the **Select the multipart attachment processing options** field. • **multipart/mixed**: Send an XML or JSON payload type with an attachment. For example, send a PDF document for review as a link in an email. • **multipart/form-data**: Send an XML or JSON payload type with an attachment. For example, you create an HTML form to upload and send an image. In the HTML form, the `method` is defined as `post` and the `enctype` (encoding type) is defined as `multipart/form-data`.

REST Adapter Trigger Response Header Page

Enter the REST Adapter response header properties for this endpoint.

> **Note:**
>
> If you specify a custom header name that is the same as a standard header name, an error occurs. Ensure that you specify unique names for your custom headers.

Specify the standard HTTP response headers to use.

Element	Description
Add Standard Response Headers	Select the standard HTTP response header to use from the default dropdown list. • Click the **Add** icon to add an additional row, then select the standard HTTP response header to use from the dropdown list. Standard headers available for selection include, but are not limited to, the following: – **Content-Type**: Enables you to assign the media type of choice to the response content. The response content type is not used for processing and is sent as part of the response during postprocessing. – **Retry After**: You can send this header as part of the response back to the caller. • Click the **Delete** icon to delete the row of a selected standard HTTP response header.
HTTP Header Name	Perform the following tasks: • From the list, select the header to use.

Specify the custom HTTP response headers to use.

Element	Description
Add Custom Response Headers	Perform the following custom response header tasks: • Click the **Add** icon to add custom HTTP response headers and optional descriptions. • Click the **Delete** icon to delete the selected custom HTTP response headers.
Custom Header Name	Enter the custom header name.
Custom Header Description	Enter an optional description.

REST Adapter Trigger Operations Page

Review or edit existing operations or add a new operation. Each operation represents a different pick action branch in a single integration. The maximum number of operations (branches) you can create in one integration is six. Each entry point can be configured with a different resource URI and HTTP action/verb, as necessary. This feature eliminates the need to create multiple integrations (each with a separate resource URI and verb) to perform different operations. You can expose multiple entry points to a single integration with a pick action that uses the REST Adapter as the trigger connection.

See Receive Requests for Multiple Resources in a Single REST Adapter Trigger Connection of *Using Integrations in Oracle Integration 3*.

Element	Description
Operation	Displays the operation name entered on the Resource Configuration page.
Resource	Displays the endpoint relative resource URL selected on the Resource Configuration page.
HTTP Method	Displays the action selected on the Resource Configuration page.
Edit/Delete	Select to edit or delete an operation and its endpoint relative resource URL and action.

Element	Description
Add another operation	Select to return to the Resource Configuration page to add another operation name, endpoint relative resource URL, and action.

REST Adapter Trigger Operation Selection Page

Enter the REST Adapter invoke operation selection parameters for this endpoint.

Element	Description
Business Object	Select the business object (resource) to use in this connection.
Operations	Select the operation (method) to perform on the business object in this connection.

Summary Page

You can review the specified adapter configuration values on the Summary page.

Element	Description
Summary	Displays a summary of the configuration values you defined on previous pages of the wizard. The information that is displayed can vary by adapter. For some adapters, the selected business objects and operation name are displayed. For adapters for which a generated XSD file is provided, click the XSD link to view a read-only version of the file. To return to a previous page to update any values, click the appropriate tab in the left panel or click **Go back**. To cancel your configuration details, click **Cancel**. Click **generate a sample cURL** to generate sample `cURL` syntax for the configuration options that you have selected during REST Adapter connection configuration, such as security policy, headers, parameters, and so on.

Add the REST Adapter as an Invoke Connection

When you drag the REST Adapter into the integration canvas as an invoke connection, the Adapter Endpoint Configuration Wizard is invoked. Based on your selections in the wizard, the following pages can be displayed.

Topics:

- REST Adapter Invoke Basic Information Page
- REST Adapter Invoke Request Parameters Page
- REST Adapter Invoke Request Page
- REST Adapter Invoke Request Headers Page
- REST Adapter Invoke Response Page
- REST Adapter Invoke Response Header Page

- REST Adapter Invoke Operation Selection Page
- Summary Page

REST Adapter Invoke Basic Information Page

Enter the REST Adapter user name, description, relative resource URI, and endpoint action. You can also select to add query and template parameters or configure a request and/or response for the endpoint.

Element	Description
What do you want to call your endpoint?	Provide a meaningful name so that others can understand the connection. For example, if you are creating a source Oracle REST connection, you may want to name it `ExposeFlowAsRESTResource`. You can include English alphabetic characters, numbers, underscores, and dashes in the name. You cannot include the following: • Blank spaces (for example, `My REST Connection`) • Special characters (for example, `#;83& or res(t)4`) • Multibyte characters
What does this endpoint do?	Enter an optional description of the connection's responsibilities (for example,`This inbound REST connection exposes this integration flow as a REST resource`).

What is the endpoint's relative resource URI?	Specify the relative path associated with the resource. The path can contain template parameters specified with curly braces (for example, `{order-id}`). A resource is any source of specific information that can be addressed. The resource path follows a fixed, prefixed URL appended with the specified relative path. By default, the URL is prefixed with the following path:
	`http://`*`host`*`:`*`port`*`/integration/flowapi/rest/`*`INTEGRATION_NAME`*
	For example, if the integration name is `ExposeFlowAsRESTResource`, the URL becomes:
	`http://`*`host`*`:port/integration/flowapi/rest/EXPOSEFLOWASRESTRESOURCE`
	You can override the URL, except for the fixed part at the beginning:
	`host``:`*`port`*`/integrations`

Element	Description
What action do you want to perform on the endpoint?	Select a single HTTP action (method) for the endpoint to perform: • **GET**: Retrieves (reads) information (for example, makes queries). If you select this option, you cannot configure a request payload for this endpoint. • **PUT**: Updates information. • **POST**: Creates information. • **DELETE**: Deletes information. If you select this option, you cannot configure a request payload for this endpoint. **PATCH**: Partially updates existing resources (for example, when you only need to update one attribute of the resource). **Note**: The **PATCH** verb does not work with a non-SSL REST service.
Select any options that you want to configure	Select the options that you want to configure: • **Add and review parameters for this endpoint**: Click to specify the query parameters and view the template request parameters created as part of the resource URI for this endpoint. If you select this option and click **Next**, the Request Parameters page is displayed. • **Configure a request payload for this endpoint**: Click to configure the request payload for this endpoint, including specifying the schema location and payload type with which you want the endpoint to reply. You can also select this option if you want to include an attachment with the inbound request. If you select this option and click **Next**, the Request page is displayed. • **Configure this endpoint to receive the response**: Click to configure the response payload for this endpoint, including specifying the schema location and payload type that you want the endpoint to receive. If you select this option and click **Next**, the Response page is displayed.
Configure Request Headers?	Select the type of request header to configure: • **Standard**: Select to configure standard HTTP headers for the request message. • **Custom**: Select to configure custom HTTP headers for the request message.
Configure Response Headers?	Select the type of response header to configure: • **Standard**: Select to configure standard HTTP headers for the response message. • **Custom**: Select to configure custom HTTP headers for the response message.

REST Adapter Invoke Request Parameters Page

Enter the REST Adapter request parameters for this endpoint.

Element	Description
Resource URI	Displays the endpoint relative resource URI entered on the Basic Info page.
HTTP Method	Displays the action to perform on the endpoint that you selected on the Basic Info page.
Specify Query Parameters	Specify query parameters for the REST endpoint. Click the **Add** icon to display a row for entering the parameter name and selecting its data type. For example, specify `state` and select a data type of **string**. Click the **Delete** icon to delete a selected row.
Template Parameters	Displays the template parameters in the relative resource URI. Template parameters are based on details you specified on the Basic Info page and cannot be edited. Template parameters must be defined as part of a path with curly braces around them. For example, the URL `default/customers/{cust-id}/{ship-id` has `cust-id` and `ship-id` template parameters. You can change the data type for the parameters. **Note**: • Any query and template parameters added or configured are available for mapping in the mapper and in the actions in integrations. • Query and template parameter values added in the URL specified on the Connection page do not appear in the mapper. Instead, the template and query parameters must be configured in the Adapter Endpoint Configuration Wizard for those parameters to appear in the mapper.

REST Adapter Invoke Request Page

Enter the REST Adapter request payload details for the endpoint.

Element	Description
Select the multipart attachment processing options	• **Request is multipart with payload**: Select to send multipart attachments as part of the request along with JSON or XML content as the payload request. • **Multipart request is of type multipart/form-data with HTML form payload**: Select to send multipart attachments as part of the request along with HTML form as the payload request. You must first select the **Response is multipart with payload** option before you can select this option. This selection assumes that the media type is multipart/form-data.

Element	Description
Select the request payload format	**Note**: • Ensure that the sample JSON or the uploaded XML schema is representative of the actual runtime messages exchanged with the endpoint. A mismatch in the structure or type of runtime messages can result in errors. • If you upload a schema file without a target namespace, a surrogate namespace is added to the schema file that all messages then use: `http://xmlns.oracle.com/cloud/adapter/nxsd/surrogate` Select the request payload format to use. The request payload body must be defined by the XSD element that defines the structure of this representation. • **XML Schema** • **JSON Sample**: Select this option to use Swagger and RAML files. JSON sample files of up to 100 KB in size are supported. Empty arrays in JSON sample files are not supported. For information, see Empty Arrays Are Not Supported in Sample JSON Files. You may need to process large JSON sample files with special characters before using the Adapter Endpoint Configuration Wizard. See Large Sample JSON File Processing with Special Characters. • **XML Sample (Single or No Namespace)**: Select this option to use an XML document to generate the schema. • **Binary**: Use with payloads that are unstructured and inline — for example, `application/octet-stream`. It preserves the file contents, but requires the receiver to determine file type, for example, from the filename extension. The Internet media type for an arbitrary byte stream is `application/octet-stream`. A list of commonly used types is shown in a dropdown list. You can select a type from this list or provide a type not listed by selecting **Other Media Type** and entering the type in the text box. **Note:** Binary payload support is only available when the adapter is used as an invoke, not a trigger.
Schema Location	Specify the schema file in either of the following ways: • Click **Browse** to select the request schema file to use. • Click **<<inline>>** to copy and paste the JSON payload or URL into a text field. Click **OK** when complete.
Element	Select the element that defines the payload structure. This field is not displayed until you import the request payload file. Once you browse for and select the schema or JSON sample file, the schema is displayed automatically. It also displays a combination box that selects the root element by default.

Element	Description
What is the media-type of the Request Body? (Content-Type Header)	• **XML**: Displays the payload in XML format. • **XML (text)**: Displays the payload in XML text format. • **JSON**: Displays the payload in JavaScript Object Notation (JSON) format. • **URL-encoded**: Displays the payload in URL-encoded format. • **Other Media Type**: Select to display the payload in another format (for example, `application/oracle.cloud+json`). You can only specify the media types that end with `+json` or `+xml`. The following media types are supported implicitly and cannot be configured. At runtime, the request media type is in the form of an `http Content-Type` header. The expected response media type is specified through an `Accept` header. Any service can be accessed through either of these media types. – Application/XML – Application/JSON Select the multipart attachment type for the endpoint to receive. This field is displayed if you selected the **Request is multipart with payload** option. • **multipart/mixed**: Send an XML or JSON payload type with an attachment. For example, send a PDF document for review as a link in an email. • **multipart/form-data**: Send an XML or JSON payload type with an attachment. For example, you create an HTML form to upload and send an image. In the HTML form, the `method` is defined as `post` and the `enctype` (encoding type) is defined as `multipart/form-data`. You can also send the attachment alone without a payload when using this attachment type.
Send Query Parameter as form-data in message body	Select if you want to pass URL-encoded form data in the payload. The values are derived from the query parameters you defined on the Request Parameters page. However, instead of submitting the query parameters, they are sent as form data in the message body with this option selected. This field is only displayed if you made the following selections in the Adapter Endpoint Configuration Wizard: • The **POST** verb was selected on the Basic Info page. • The **Configure a request payload for this endpoint** and **Add and review parameters for this endpoint** options were selected on the Basic Info page. • Query parameters were specified on the Request Parameters page.

REST Adapter Invoke Request Headers Page

Enter the REST Adapter request header properties for this endpoint.

> **Note:**
>
> If you specify a custom header name that is the same as a standard header name, an error occurs. Ensure that you specify unique names for your custom headers.

Specify the standard HTTP request headers to use.

Element	Description
Add Standard Request Headers	Select the standard HTTP request header to use from the default dropdown list. • Click the **Add** icon to add an additional row, then select the standard HTTP request header to use from the dropdown list. Standard headers available for selection include, but are not limited to, the following: – **Accept**: Response processing occurs according to the static contract defined at design time. The dynamic value overrides the value sent to the endpoint. The response returned must match the **Accept** header configured at design time. The overridden header must be a variation of the content length specified at design time. – **Authorization**: The dynamic header value overrides any authorization performed as part of the security policy. Track any authorization failures to the given header property. – **Content Length**: The REST Adapter always infers the content length from the actual content length passed to the target endpoint. If there is no content, but the target endpoint expects the content-length header with a value of `0`, then as with all request headers, you can include the standard HTTP request header from the Adapter Endpoint Configuration Wizard and assign it a value of `0` in the mapper. In Oracle Integration 3, a Content-Length header with a value of `0` is only sent for PUT and PATCH verbs with no content. – **Content-Type**: The dynamic header is not passed to the translation framework and translation continues according to the static configuration at design time. At runtime (post-message processing), the dynamic header value is sent to the endpoint. – **Origin**: You can set the origin request header as part of the outgoing HTTP request. • Click the **Delete** icon to delete the row of a selected standard HTTP request header.
HTTP Header Name	Perform the following tasks: • From the list, select the header to use.

Specify the custom HTTP request headers to use.

Element	Description
Add Custom Request Headers	Perform the following custom request header tasks: • Click the **Add** icon to add custom HTTP request headers and optional descriptions. • Click the **Delete** icon to delete the selected custom HTTP request headers.
Custom Header Name	Enter the custom header name.
Custom Header Description	Enter an optional description.

REST Adapter Invoke Response Page

Enter the REST Adapter response payload details for the endpoint.

Element	Description
Resource URI	Displays the endpoint relative resource URI entered on the Basic Info page.
HTTP Method	Displays the action to perform on the endpoint that you selected on the Basic Info page.
Select the multipart attachment processing options	Configure the following options based on whether the request is inbound or outbound. For inbound (trigger) responses, select the multipart attachment type to include. • **Response is multipart with payload**: Select to receive the response from the payload. • **Multipart response is of type multipart/form-data with HTML form payload**: Select for the REST endpoint to accept to configure an HTML form. You must first select the **Response is multipart with payload** option before you can select this option. This selection assumes that the media type is multipart/form-data.

Element	Description
Select the response payload format	**Note**: • Ensure that the sample JSON or the uploaded XML schema is representative of the actual runtime messages exchanged with the endpoint. A mismatch in the structure or type of runtime messages can result in errors. • If you upload a schema file without a target namespace, a surrogate namespace is added to the schema file that all messages then use: `http://xmlns.oracle.com/cloud/adapter/nxsd/surrogate` Select the response payload format to use. The response payload body must be defined by the XSD element that defines the structure of this representation. • **XML Schema** • **JSON Sample**: Select this option to use Swagger and RAML files. JSON sample files of up to 100 KB in size are supported. Empty arrays in JSON sample files are not supported. For information, see Empty Arrays Are Not Supported in Sample JSON Files. You may need to process large JSON sample files with special characters before using the Adapter Endpoint Configuration Wizard. See Large Sample JSON File Processing with Special Characters. • **XML Sample (Single or No Namespace)**: Select this option to use an XML document to generate the schema. • **Binary**: Use with payloads that are unstructured and inline — for example, `application/octet-stream`. It preserves the file contents, but requires the receiver to determine the file type, for example, from the filename extension. The Internet media type for an arbitrary byte stream is `application/octet-stream`.
Schema Location	Specify the schema file in either of the following ways: • Click **Browse** to select the response schema file to use. • Click **<<inline>>** to copy and paste the JSON payload or URL into a text field. Click **OK** when complete.
Element	Select the element that defines the payload structure. This field is not displayed until you import the response payload file. Once you browse for and select the schema file, it displays a combination box that selects the root element by default.

Element	Description
What is the media-type of the Response Body? (Accept Header)	Select the payload type with which you want the endpoint to reply. • **XML**: Displays the payload in XML format. • **XML (text)**: Displays the payload in XML text. • **JSON**: Displays the payload in JavaScript Object Notation (JSON) format. • **Other Media Type**: Select to display the payload in another format (for example, `application/oracle.cloud+json`). You can only specify media types that end with `+json` or `+xml`. The following media types are supported implicitly and cannot be configured. At runtime, the request media type is in the form of an `http Content-Type` header. The expected response media type is specified through an `Accept` header. Any service can be accessed through either of these media types. – Application/XML – Application/JSON Select the multipart attachment type for the endpoint to receive. This field is displayed if you selected the **Response is multipart with payload** option. • **multipart/mixed**: Send an XML or JSON payload type with an attachment. For example, send a PDF document for review as a link in an email. • **multipart/form-data**: Send an XML or JSON payload type with an attachment. For example, you create an HTML form to upload and send an image. In the HTML form, the `method` is defined as `post` and the `enctype` (encoding type) is defined as `multipart/form-data`.

REST Adapter Invoke Response Header Page

Enter the REST Adapter response header properties for this endpoint.

> **Note:**
>
> If you specify a custom header name that is the same as a standard header name, an error occurs. Ensure that you specify unique names for your custom headers.

Specify the standard HTTP response headers to use.

Element	Description
Add Standard Response Headers	Select the standard HTTP response header to use from the default dropdown list. • Click the **Add** icon to add an additional row, then select the standard HTTP response header to use from the dropdown list. Standard headers available for selection include, but are not limited to, the following: – **Content Length**: The content length of the response is propagated to Oracle Integration (regardless of the length). The response may have been translated and the actual values may no longer match. This is the response header corresponding to the original message. – **Content-Type**: The response header is propagated along with the response to the integration. – **Retry-After**: The response header sent by a target endpoint is returned to the integration. • Click the **Delete** icon to delete the row of a selected standard HTTP response header.
HTTP Header Name	Perform the following tasks: • From the list, select the header to use.

Specify the custom HTTP response headers to use.

Element	Description
Add Custom Response Headers	Perform the following custom response header tasks: • Click the **Add** icon to add custom HTTP response headers and optional descriptions. • Click the **Delete** icon to delete the selected custom HTTP response headers.
Custom Header Name	Enter the custom header name.
Custom Header Description	Enter an optional description.

REST Adapter Invoke Operation Selection Page

Enter the REST Adapter invoke operation selection parameters for this endpoint.

Element	Description
Business Object	Select the business object (resource) to use in this connection.
Operations	Select the operation (method) to perform on the business object in this connection.

Summary Page

You can review the specified adapter configuration values on the Summary page.

Element	Description
Summary	Displays a summary of the configuration values you defined on previous pages of the wizard. The information that is displayed can vary by adapter. For some adapters, the selected business objects and operation name are displayed. For adapters for which a generated XSD file is provided, click the XSD link to view a read-only version of the file. To return to a previous page to update any values, click the appropriate tab in the left panel or click **Go back**. To cancel your configuration details, click **Cancel**. Click **generate a sample cURL** to generate sample `cURL` syntax for the configuration options that you have selected during REST Adapter connection configuration, such as security policy, headers, parameters, and so on.

5

Implement Common Patterns Using the REST Adapter

You can use the REST Adapter to implement the following common patterns.

Differences Between Implementation Patterns and Recipes

Unclear about the difference between implementation patterns and recipes? Both assets provide different solutions.

- Implementation patterns describe a common use for this adapter. The pattern can range from a simple configuration task (such as how to configure a specific security policy for this adapter) to a high level overview of how to use this adapter in an integration.
- Recipes, known as *prebuilt integrations*, are preassembled, easily installable, integration solutions. A recipe contains all the resources required for a specific integration scenario. The resources include integration flows, adapter connections, lookups, and certificates. Use a recipe to quickly get started building an integration. See the Recipes and Accelerators page on the Oracle Help Center.

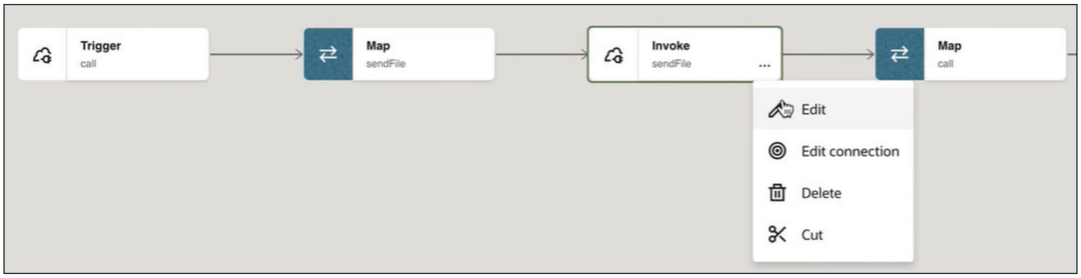
Topics:

- Create a Keystore File for a Two-Way, SSL-Based Integration
- Access Oracle Cloud Infrastructure Service Resources Using RPST
- Invoke a Service Provider API with a JWT Assertion
- Propagate User Identity from the REST Adapter to Oracle NetSuite with Custom JWT Claims

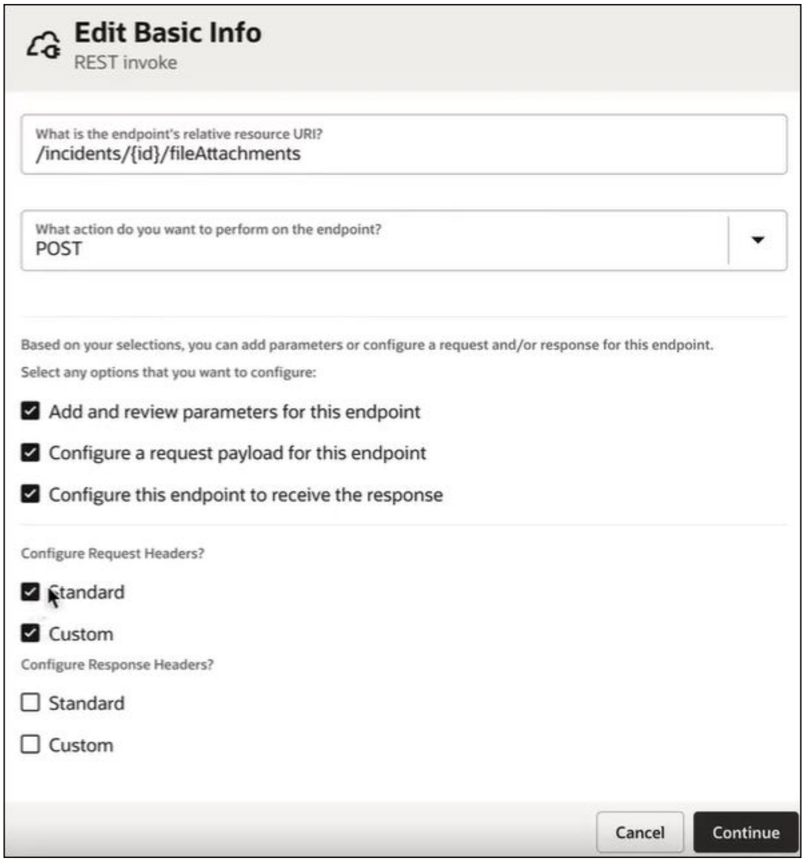
Connect to an Endpoint that Requires a Content-Length Header to Be Sent

In Oracle Integration 3, invoke connection requests sent to an endpoint default to chunked encoding. This means no Content-Length header is sent. If your endpoint requires a Content-Length header, the HTTP response returns a `400` or `411` error. Perform the following steps to enable a Content-Length header to be sent.

1. Edit the Adapter Endpoint Configuration Wizard for the invoke connection. For example:



2. On the Basic Info page, select the **Standard** request header check box.



Edit Basic Info
REST invoke

What is the endpoint's relative resource URI?
/incidents/{id}/fileAttachments

What action do you want to perform on the endpoint?
POST

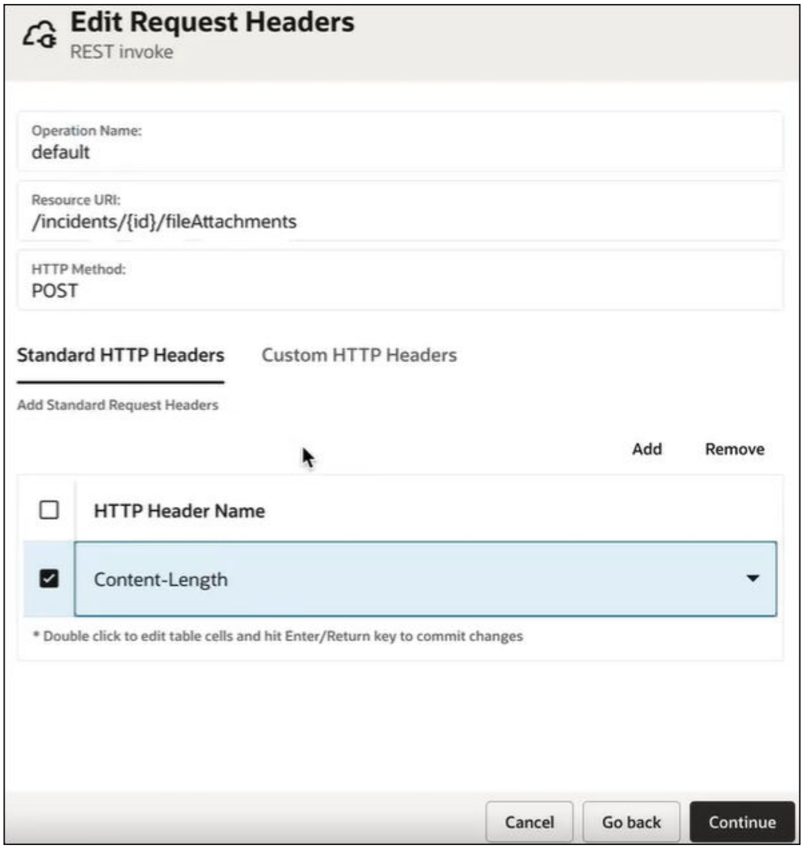
Based on your selections, you can add parameters or configure a request and/or response for this endpoint.
Select any options that you want to configure:

☑ Add and review parameters for this endpoint
☑ Configure a request payload for this endpoint
☑ Configure this endpoint to receive the response

Configure Request Headers?
☑ Standard
☑ Custom

Configure Response Headers?
□ Standard
□ Custom

3. Click through the Adapter Endpoint Configuration Wizard without making changes until you reach the Edit Request Headers page.
4. In the **Standard HTTP Headers** section, click **Add** and select **Content-Length** from the list.



Edit Request Headers
REST invoke

Operation Name: default

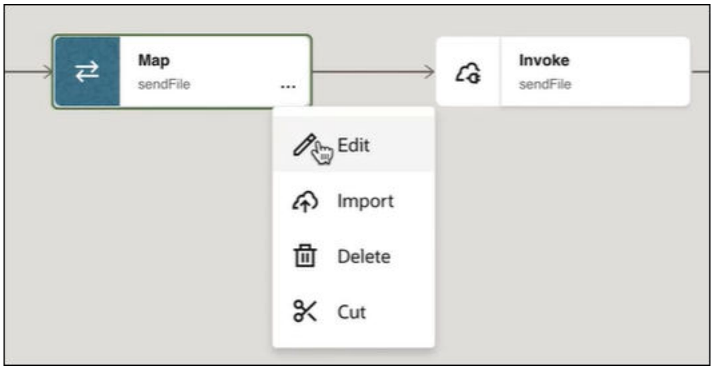
Resource URI: /incidents/{id}/fileAttachments

HTTP Method: POST

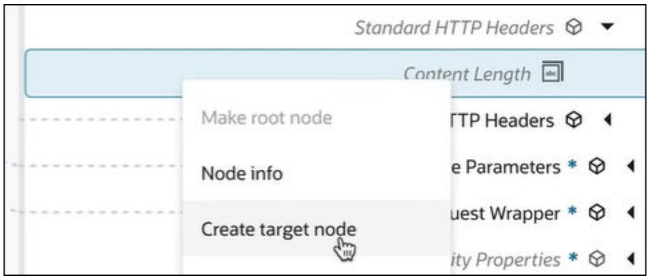
☐	HTTP Header Name
☑	Content-Length

* Double click to edit table cells and hit Enter/Return key to commit changes

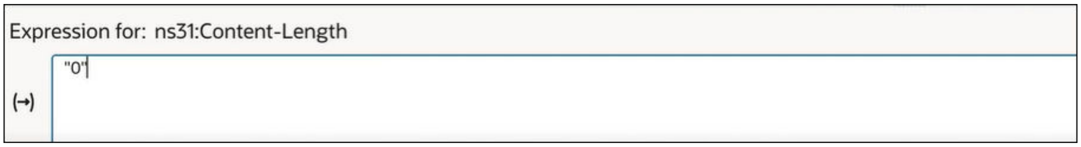
5. Click through the remaining pages of the Adapter Endpoint Configuration Wizard without making changes.
6. On the Summary page, click **Finish**.
7. Open the mapper in front of the invoke connection for editing.



8. In the **Target** section, right-click **Content Length** under **Standard HTTP Headers**.
9. Select **Create target node**.



10. In the Expression Builder at the bottom, set the value to `"0"`. This value is not significant. It simply enables the REST Adapter to understand that a Content-Length header is required.


Expression for: ns31:Content-Length

"0"

11. Exit the mapper and save the integration.
12. Run the integration.

    This enables a Content-Length header to be sent to the endpoint.

OAuth-Protected Patterns

You can use the REST Adapter to implement the following common patterns using OAuth protection.

- Configure the REST Adapter to Consume a REST API Protected with OAuth Custom Two Legged Token-Based Authentication
- Configure the REST Adapter to Consume a REST API Protected with OAuth Custom Three Legged Flow Token-Based Authentication
- Configure the REST Adapter to Consume a REST API Protected with OAuth 1.0 One-Legged Authentication
- Allow Client Applications to Consume an Integration Exposed as an OAuth-Protected REST API

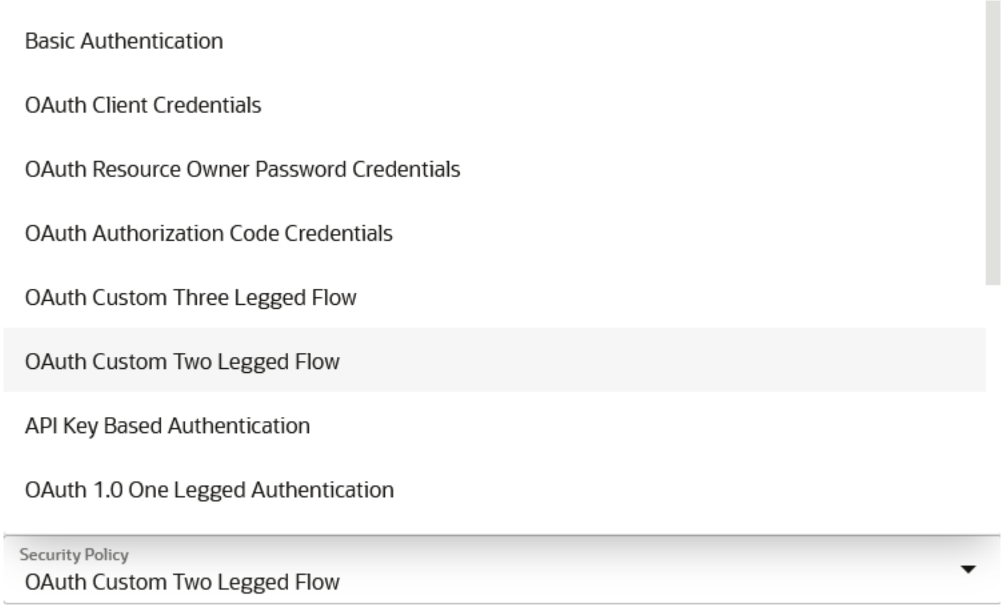
Configure the REST Adapter to Consume a REST API Protected with OAuth Custom Two Legged Token-Based Authentication

This section provides an overview of the OAuth Custom Two Legged Flow security policy. This policy is useful when the Basic Authentication security policy is not sufficient.

Most HTTP services typically use the OAuth authorization framework to protect their resources. In accordance with the OAuth 2.0 specification, the OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service or by enabling the third-party application to obtain access on its own behalf.

The REST Adapter enables you to integrate with any REST-enabled service including OAuth services. To interact with an OAuth endpoint, you must create a one-time reusable connection on the Connections page of Oracle Integration. Configure the connection with the base URI and security configuration.

The following security policy options are available on the Connections page for the REST Adapter.



Basic Authentication

OAuth Client Credentials

OAuth Resource Owner Password Credentials

OAuth Authorization Code Credentials

OAuth Custom Three Legged Flow

OAuth Custom Two Legged Flow

API Key Based Authentication

OAuth 1.0 One Legged Authentication

Security Policy
OAuth Custom Two Legged Flow

Each option is applicable in a different context and is used to negotiate and obtain a valid access token. Read your REST service provider documentation to identify the applicable policy.

The following section describes a flexible OAuth security policy that can be used in OAuth custom two legged flows called as an **OAuth Custom Two Legged Flow**.

OAuth 2.0 specification defines the following OAuth flows:

- OAuth Client Credentials
- OAuth Resource Owner Password Credentials
- OAuth Authorization Code Credentials
- OAuth Implicit Grant Authorization

The **OAuth Client Credentials** and **OAuth Resource Owner Password Credentials** options are categorized as OAuth custom two legged flows because the client application directly obtains access on its own without the resource owner’s intervention.

An HTTP request is typically sent to the authorization server passing the client application credentials (note that these are different from the resource owner credentials and can be obtained by registering the client application with the authorization server), the grant type and scope, and other required properties. The authorization server responds to this request by sending an access token, optionally with a token type, an expiry, and sometimes a refresh token.

The following example describes a sample access token request with Twitter (a popular microblogging site that supports OAuth2). For more information about Twitter developer documentation, visit https://dev.twitter.com/oauth/application-only.

```
POST /oauth2/token HTTP/1.1
Host: api.twitter.com
Content-Type: application/x-www-form-urlencoded
```

```
Authorization: Basic a3NmM1yRnFweAx==

grant_type=client_credentials
```

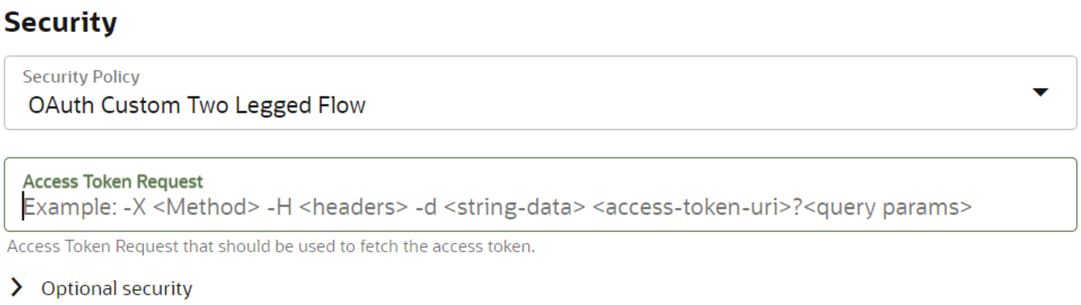
According to the Twitter developer documentation, this request is required to obtain an access token from Twitter. An HTTP basic authentication header is created by using the client ID and client secret.

If the request is formatted correctly, the server responds with a JSON-encoded payload. This is fairly straight forward.

```
{"token_type":"bearer","access_token":"AAAAAAAAAA"}
```

The following steps describe the **OAuth Custom Two Legged Flow** security policy and each field in the context of this scenario.

Step 1: Configure the Access Token Request



Security

Security Policy
OAuth Custom Two Legged Flow

Access Token Request
Example: -X <Method> -H <headers> -d <string-data> <access-token-uri>?<query params>

Access Token Request that should be used to fetch the access token.

The **Access Token Request** field is formed using the URI syntax of the HTTP request used to fetch the access token. The URI syntax resembles `cURL` but is more basic and only supports the following options.

Option	Value	Description	Required
`-X`	`GET	PUT	POST`	The HTTP verb in the access token request.	Yes
`-H`	`-H “key: value”`	Add each header key value pair as described. There can be multiple headers.	No
`-d`	`-d ‘data-as-string’`	String data enclosed within single quotes. Escape any quotes within the data string.	No
`URI`	`Uri` (within quotes)	- -	Yes

Parameters specified with the `-d` option should be URL-encoded. For example, assume `client_id` is the following value:

```
qwerty&r=123=&q=asdf
```

You are required to URL-encode this value using a URL encoder tool. The parameter must be URL-encoded before composing the -d data. This applies to client_secret and also a scope or any other additional value you want to put into the overall -d parameter. For example:

```
client_id = "qwerty&r=123=&q=asdf"


client_secret = "zxcvb&q=12345&=7890"
```

Access token request:

```
-X POST -H 'Content-Type: application/x-www-form-urlencoded' -d
'client_secret=zxcvb%26q%3D12345%26%3D7890&grant_type=authorization_code&redir
ect_uri=${redirect_uri}&client_id=qwerty%26r%3D123%3D%26q%3Dasdf'
https://webhook.site/44ffa856-9459-4bb5-b8db-c0ed0d3b227f
```

If data must be in a query parameter:

```
-X POST -H 'Content-Type: application/x-www-form-urlencoded'
'https://webhook.site/44ffa856-9459-4bb5-b8db-c0ed0d3b227f?
client_secret=zxcvb%26q%3D12345%26%3D7890&grant_type=authorization_code&redire
ct_uri=${redirect_uri}&client_id=qwerty%26r%3D123%3D%26q%3Dasdf'
```

Multiple -d options in the OAuth custom two-legged flow security policy can be compressed into a single -d as follows:

```
-d "grant_type=client_credentials&client_id=123"
```

Note:

- Other curl options are not supported.
- The easiest way to build this request is to use a free tool such as postman to build and validate the HTTP request to obtain an access token and then use the Generate Code Snippet/Code option to get curl syntax. Remove the curl from the beginning to get the URI syntax. The following example shows URI syntax:

  ```
  -X POST -H "Content-Type: application/x-www-form-urlencoded" -H
  "Authorization:
  Basic a3NmM0J6czJG==" -d 'grant_type=client_credentials'
  https://api.twitter.com/oauth2/token
  ```

The URI syntax allows you to control the access token request. The following is a typical access token response.

```
{
  "access_token": "1-253912-240049694-f85c1d679211c",
  "expires_in": 21599,
  "token_type": "Bearer",
```

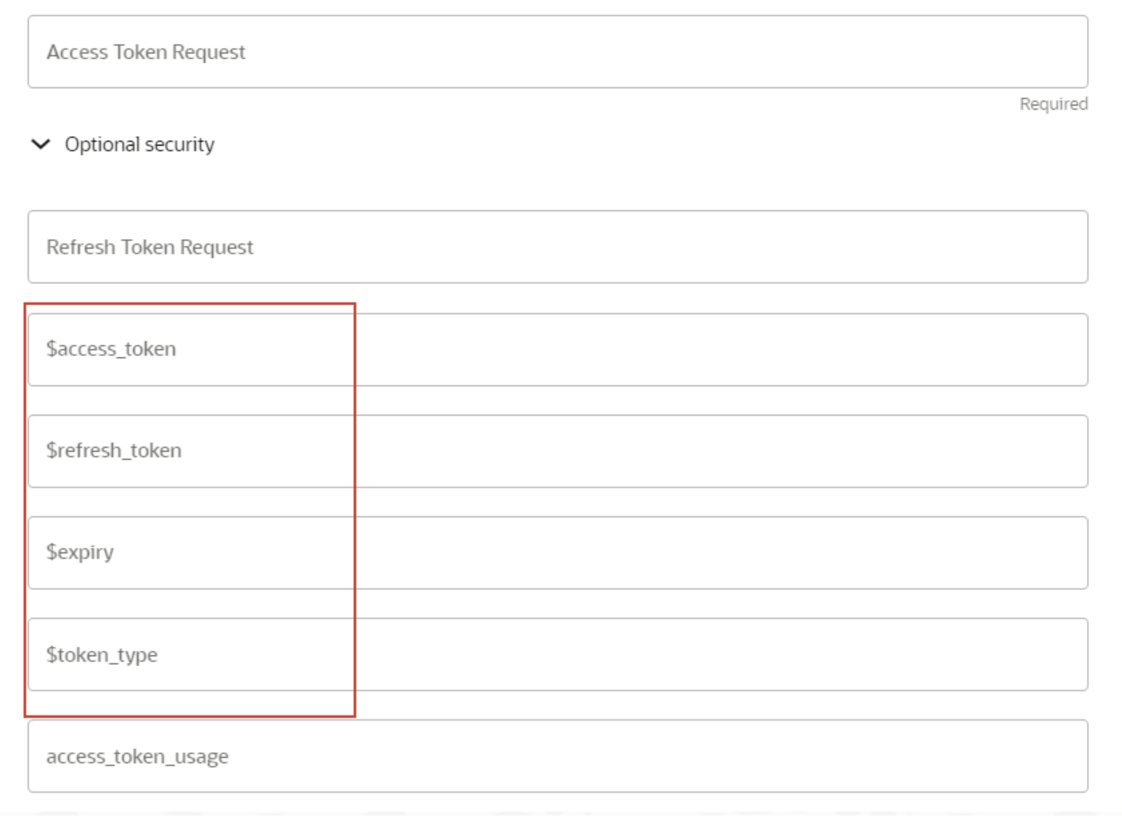
```
  "refresh_token": "5707efdf04912f53b61cb5ec5dc7f166"
}
```

Step 2: Parse and Extract Tokens from Access Token Response

Note:

Skip this step if the access token response has properties as highlighted previously.

If the request is good, the authorization server returns an HTTP response with a success status. The response contains the access token and may also contain several operational details about the token such as the type of the token, its expiry, and refresh token as described previously.



Access Token Request

Required

Optional security

Refresh Token Request

$access_token

$refresh_token

$expiry

$token_type

access_token_usage

By default, the `$variables` are mapped to property names containing relevant tokens as follows:

Property Name	Default Mapping to a Property with Name	Example Property Name
$access_token	`access.[tT]oken`	`access_token`
$refresh_token	`refresh.[tT]oken`	`refresh_token`
$expiry	`expires_in`	`expires_in`
$token_type	`token.?[tT]ype`	`token_type`

The default values match the sample response. Therefore, this step is not required and can be skipped.

However, if the access token response is not standard, then you must define rules to fetch tokens from the access token response.

For example, assume the access token response is as follows:

```
{   "access_token": "1-253912-240049694-f85c1d679211c", "expiry": 21599,
"token_type": "Bearer",   "extended_token":
"5707efdf04912f53b61cb5ec5dc7f166" }
```

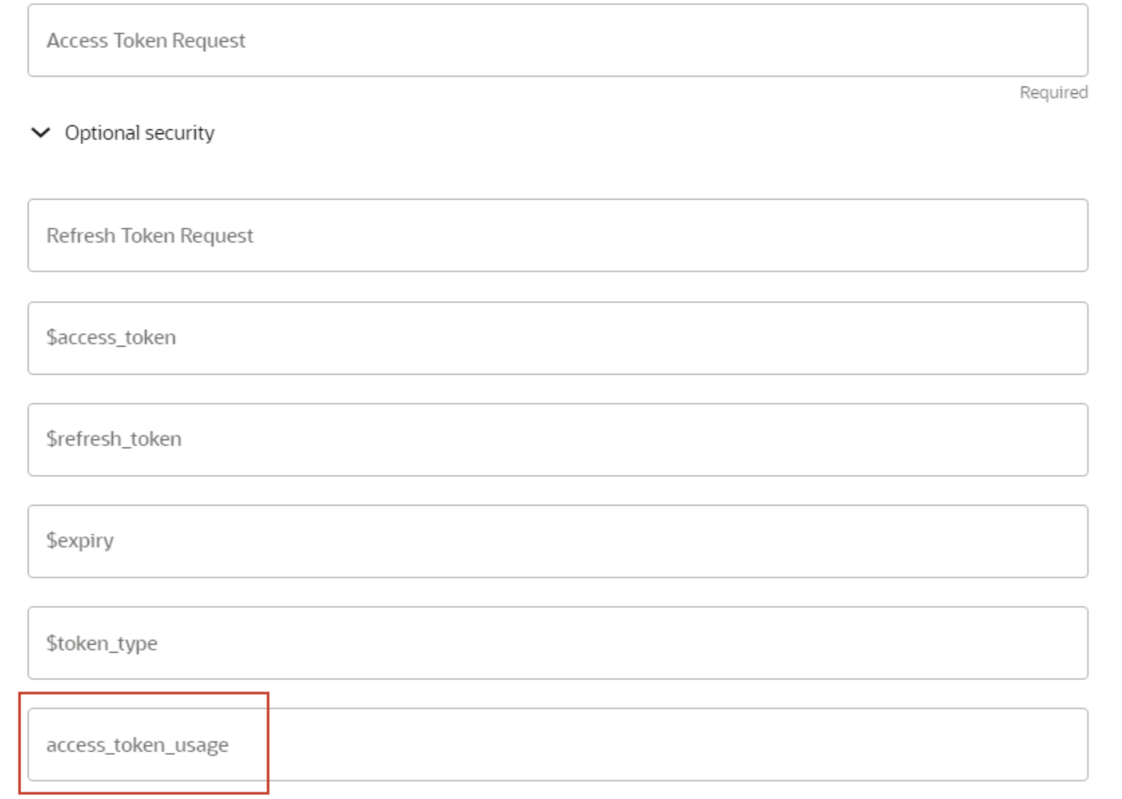
In this case, the authorization server returns a response, but chooses to specify the expiry and the refresh token differently. This step is required to map these properties to the variables.

Variable Name	Default Mapping to a Property with Name	Example Property Name
$refresh_token	`extended_token`	`extended_token`
$expiry	`Expiry`	`Expiry`

Variables can be used in the configuration using the `${variable}` syntax once a value has been assigned. For example, `$access_token` is assigned a value after an access token request is made. The value of this variable may be useful while specifying the `access_token` usage or the `refresh_token_request` later.

Step 3: Access Token Usage (Important)

Access token usage describes how to pass the access token to access a resource. Enter this information carefully because this usage governs how Oracle Integration passes the negotiated access token to the endpoint.



Access Token Request

Required

Optional security

Refresh Token Request

$access_token

$refresh_token

$expiry

$token_type

access_token_usage

The default value for this field is:

```
-H Authorization: ${token_type} ${access_token}
```

At runtime, the values of `${token_type}` and `${access_token}` are retrieved based on the fetch rule and passed as an authorization header along with the endpoint request.

The literal value can also be used as follows:

```
-H API-Token: Bearer ${access_token}
```

Access Token Usage	Description	Example
`-H Authorization: ${token_type} ${access_token}`	The access token is passed as a header at runtime for accessing the protected resource.	`-H Authorization: Bearer AASDFADADX`
`?api_key=${access_token}`	The access token is passed as a query parameter at runtime for accessing the protected resource.	`http://someapi.com/employee?api_key=ASDFADAX`

Step 4: Refresh Token Request (Optional)

Some providers provide a mechanism to refresh a given access token. This sort of method is generally part of a resource owner password credentials (ROPC) flow. However, there have been instances where you also use this with client credentials even when the specification says otherwise.

The refresh token request typically takes the refresh token and returns a new access token as a response along with operational attributes such as the type of token, its expiry, and another refresh token.

The refresh token request must also be specified in a syntax similar to the access token request and prescribes to the same rules.

A sample refresh token request is as follows:

```
-X POST 'https://sample.com/oauth2/token?refresh_token=${refresh_token}
&client_id=[YOUR_CLEINT_ID]&client_secret=[YOUR_CLIENT_SECRET]&grant_type=refr
esh_token'
```

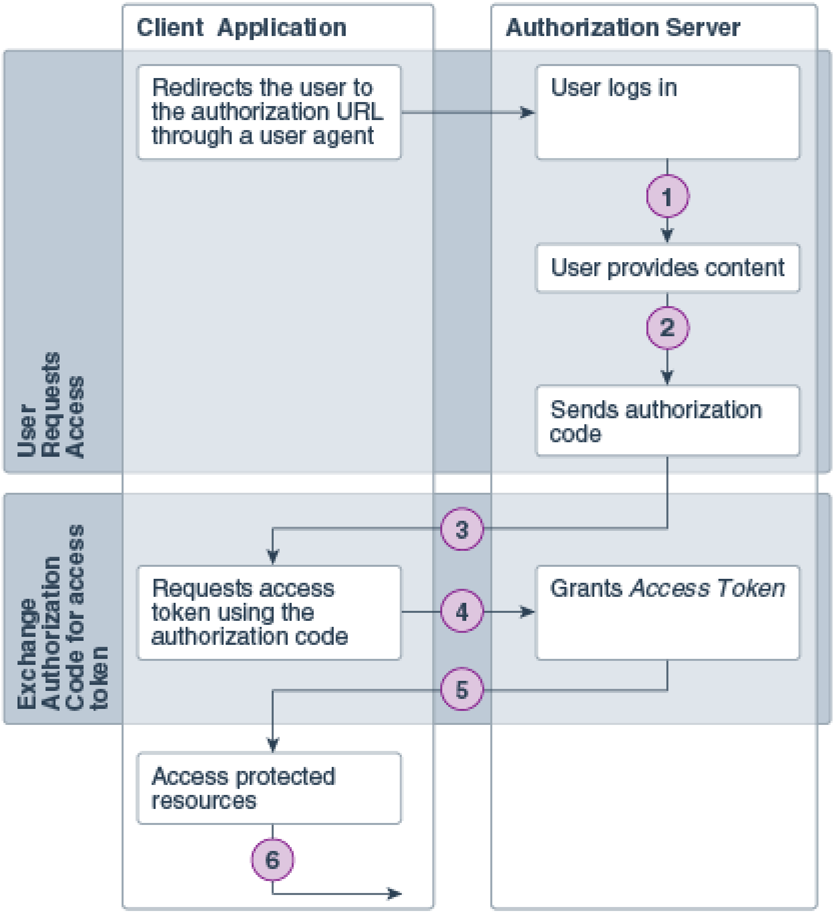
This request contains a variable that is replaced with the actual value of the current refresh token at runtime.

See Configure Connection Security.

Configure the REST Adapter to Consume a REST API Protected with OAuth Custom Three Legged Flow Token-Based Authentication

This section provides an overview of the OAuth Custom Three Legged Flow security policy.

The following steps are performed as part of a typical OAuth authorization code credentials flow.


Client Application
Authorization Server
User Requests Access
Redirects the user to the authorization URL through a user agent
User logs in
1
User provides content
2
Sends authorization code
Exchange Authorization Code for access token
3
Requests access token using the authorization code
4
Grants *Access Token*
5
Access protected resources
6

Step	Description
1	The user specifies the authorization request URI. The user is redirected by the user agent (browser) to the authorization URI.
2	The resource owner logs in to authenticate and provide consent to the client application to access its resources.
3	The authorization server sends a callback request to the client application and sends the authorization code.
4	The client application extracts the authorization code from the request and uses it to send another request to the authorization server to get an access token.
5	The authorization server responds to the access token request by sending an access token to the client application.
6	The client application uses the access token to make requests for protected resources.

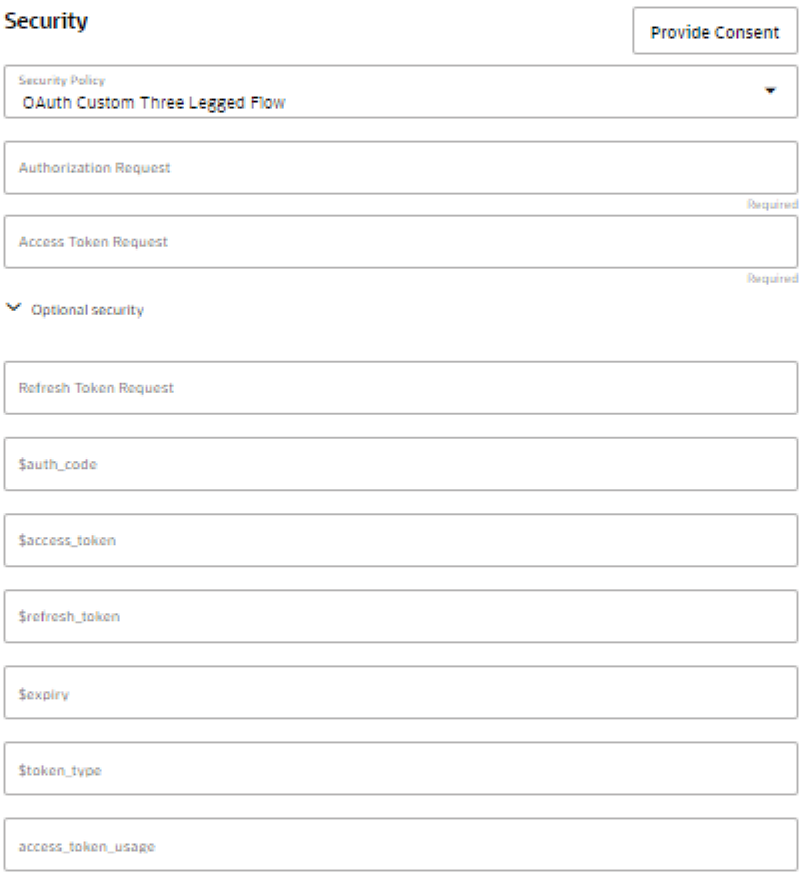
This flow is defined in the OAuth specification. However, how to perform each step in the flow is determined by the authorization server implementing the OAuth flow. There are several variations to this flow.

- The OAuth provider expects that some query parameters are passed when the user is redirected to the authorization URI.
- The provider calls the authorization code something else.
- The call for the access token should include the authorization code. However, some providers may expect it as a header or a query parameter or maybe as part of the data.

- The access token response may also wary. Some providers may return a refresh token (for example, call it extended_token or something else). Providers are known to return an expiry, whereas some providers return a JWT token, where the expiry is embedded as a claim within the token.
- Providers may also declare a custom token type.
- The call to refresh the access token may also vary from provider to provider.
- The call to access resources using the access token may also vary. Providers may expect it to be a header or a query parameter. Some providers ask the token to be passed as an authorization header. Few providers expect a custom header, and so on.

The REST Adapter provides a security policy called the **OAuth Authorization Code Credentials Flow**. This policy provides a specific implementation of the OAuth as illustrated in the OAuth specification. For all other cases, **OAuth Custom Three Legged Flow** can be used to address these customizations.

Step 1: Configure the Authorization Request



Security Policy
OAuth Custom Three Legged Flow

Specify the authorization URI where the resource owner authenticates and provides consent in the **Authorization Request** field. The client ID and scope are typically passed as query parameters with the redirect URI from where the authorization server must send a callback and the authentication code.

Oracle Integration has a fixed endpoint to receive this callback so you can specify the URI directly or pass a reference `${refresh_token}` that is automatically resolved by the platform. For example:

```
https://AUTH_URI?response_type=code&client_id=YOUR_CLIENT_ID &redirect_uri=$
{redirect_uri}&scope=app_scope
```

Step 2: Configure the Access Token Request

When the resource owner provides consent, the authorization server sends a callback to the client application along with the authorization code. The next step is for the client application to send a request for the access token using this authorization code.

If the authorization server returns the authorization code in a property named as anything but code, you must map the property name with `$auth_code`.

The access token request is used to make a call for the access token. It is supposed to send the authorization code that is not resolved until the flow is executed. Therefore, the authorization code is passed by reference as `${auth_code}` in the request.

The rules for creating the access token request remain unchanged from the **OAuth Custom Two legged Flow** option.

The **Access Token Request** value is formed using a URI syntax of the HTTP request used to fetch the access token. The URI syntax resembles `cURL`, but it is more basic and only supports the following options.

Option	Value	Description	Required
`-X`	`GET | PUT | POST`	The HTTP verb in the access token request.	Yes
`-H`	`-H “key: value”`	Add each header key value pair as described. There can be multiple headers.	No
`-d`	`-d ‘data-as-string’`	String data enclosed within single quotes. Escape any quotes within the data string.	No
`URI`	`Uri` (within quotes)	- -	Yes

Parameters specified with the `-d` option should be URL-encoded. For example, assume `client_id` is the following value:

```
qwerty&r=123=&q=asdf
```

You are required to URL-encode this value using a URL encoder tool. The parameter must be URL-encoded before composing the `-d` data. This applies to `client_secret` and also a scope or any other additional value you want to put into the overall `-d` parameter. For example:

```
client_id = "qwerty&r=123=&q=asdf"


client_secret = "zxcvb&q=12345&=7890"
```

Access token request:

```
-X POST -H 'Content-Type: application/x-www-form-urlencoded' -d
'client_secret=zxcvb%26q%3D12345%26%3D7890&grant_type=authorization_code&redir
ect_uri=${redirect_uri}&client_id=qwerty%26r%3D123%3D%26q%3Dasdf'
https://webhook.site/44ffa856-9459-4bb5-b8db-c0ed0d3b227f
```

If data must be in a query parameter:

```
-X POST -H 'Content-Type: application/x-www-form-urlencoded'
'https://webhook.site/44ffa856-9459-4bb5-b8db-c0ed0d3b227f?
client_secret=zxcvb%26q%3D12345%26%3D7890&grant_type=authorization_code&redire
ct_uri=${redirect_uri}&client_id=qwerty%26r%3D123%3D%26q%3Dasdf'
```

Multiple `-d` options in the OAuth Custom Three Legged Flow security policy can be compressed into a single `-d` as follows:

```
-d "grant_type=client_credentials&client_id=123"
```

Note:

- Other `curl` options are not supported.
- The easiest way to build this request is to use a free tool such as POSTMAN to build and validate the HTTP request to obtain an access token and then use the Generate Code Snippet/Code option to get a `cURL` syntax. Remove the `curl` from the beginning to get the URI syntax. The following example shows the URI syntax:

```
-X POST -H "Content-Type: application/x-www-form-urlencoded" -d
'false' 'https://access_token_URI?code=$
{auth_code}&client_id=YOUR_CLIENT_ID&client_secret=YOUR_CLIENT_SECR
ET&redirect_uri=${redirect_uri}&grant_type=authorization_code'
```

Step 3: Optionally Configure the Refresh Token Request

Similar to an access token request, specify the refresh token request in URI syntax, if the authorization server supports a refresh.

Step 4: Define the Fetch Rules for Intermediate Tokens

By default, the `$variables` are mapped to property names containing relevant tokens as follows:

Property Name	Default Mapping to a Property with Name	Example Property Name
$auth_code	`code`	`code`
$access_token	`access.[tT]oken`	`access_token`
$refresh_token	`refresh.[tT]oken`	`refresh_token`
$token_type	`token.?[tT]ype`	`token_type`

Property Name	Default Mapping to a Property with Name	Example Property Name
$expiry	`expires_in`	`expires_in`

This step is not required and can be skipped.

However, if the access token response is not standard, then you must define rules to fetch tokens from the access token response.

Step 5: Define the Access Token Usage (Important)

Access token usage describes how to pass the access token to access a resource. Enter this information carefully because this usage governs how Oracle Integration passes the negotiated access token to the endpoint.

See Configure Connection Security.

Configure the REST Adapter to Consume a REST API Protected with OAuth 1.0 One-Legged Authentication

This section provides an overview of the OAuth 1.0 One-Legged Authentication security policy in the Connections page. This protocol enables web sites or applications (consumers) to access protected resources from a web service (a service provider) through an API without requiring you to disclose your service provider credentials to consumers.

> **Note:**
>
> No customization is required in this policy. This is a standard OAuth policy unlike custom 2-legged and custom 3-legged OAuth policies.

You can use this security policy with service providers such as the following:

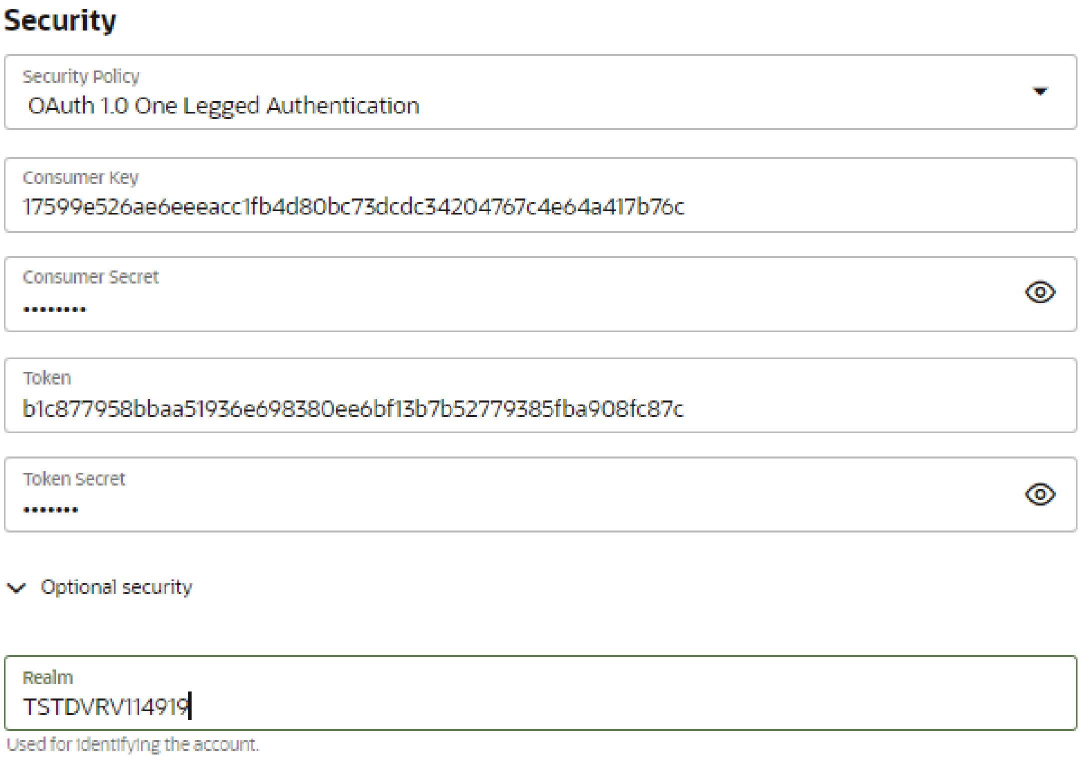
- Oracle NetSuite can expose restlets as REST APIs that are protected by OAuth 1.0 One-Legged Authentication. For example:

  ```
  https://rest.netsuite.com/app/site/hosting/restlet.nl?script=474&deploy=1
  ```

  You must be a member of Oracle NetSuite to access this restlet.

  This restlet returns a greeting in HTML.
- Twitter accounts can be protected by OAuth 1.0 One-Legged Authentication.

Configure the following fields on the Credentials dialog of the Connections page. These credentials are provided by the service provider (Oracle NetSuite or Twitter).



Security

Security Policy
OAuth 1.0 One Legged Authentication

Consumer Key
17599e526ae6eeeacc1fb4d80bc73dcdc34204767c4e64a417b76c

Consumer Secret
••••••••

Token
b1c877958bbaa51936e698380ee6bf13b7b52779385fba908fc87c

Token Secret
•••••••

Optional security

Realm
TSTDVRV114919

Used for identifying the account.

- **Consumer Key** — Specify the key that identifies the client making the request.
- **Consumer Secret** — Specify the consumer secret that authorizes the client making the request.
- **Token** — Specify the token that accesses the protected resource.
- **Token Secret** — Specify the token secret that generates the signature for the request.
- **Realm** — Specify the realm that identifies the account.

See Configure Connection Security.

Allow Client Applications to Consume an Integration Exposed as an OAuth-Protected REST API

Integrations in Oracle Integration configured using the REST Adapter as a trigger are automatically exposed as OAuth-protected REST resources. These integrations/resources can be consumed using OAuth access tokens. To access an Oracle Integration endpoint using an OAuth token, you must first acquire the token.

See Authenticate Requests for Invoking Oracle Integration Flows.

Propagate OAuth User Identity Between Services

Oracle Integration provides support for OAuth identity propagation when invoking REST API operations. OAuth identity propagation enables you to securely transfer the same user identity and access credentials across services. The services involved may use the same identity domain within Oracle Integration, a different identity domain outside Oracle Integration, or a third-party identity provider.

- How Identity Propagation in Oracle Integration Works

- Propagate User Identity

How Identity Propagation in Oracle Integration Works

It is a common business requirement to propagate the identity of a user between multiple services. For example:

- You log in to a VBCS application and call Oracle Integration, which is using the identity domain in its tenancy.
- Oracle Integration then invokes an Oracle Fusion Applications endpoint, which is using a separate identity domain in a different tenancy. The Oracle Fusion Applications endpoint must know the end user making the call to drive its business logic.
- Oracle Integration also invokes a Salesforce endpoint, using a third-party identity provider outside of Oracle. The Salesforce endpoint must also know the end user making the call to drive its business logic.

Identity propagation works as follows for these types of scenarios:

- User Authentication - A user authenticates with an identity provider using their credentials.
- Token Issuance - Upon successful authentication, the identity provider issues a JWT access token containing the user's identity information and authorized scopes.
- Token propagation - When the user accesses a different service, the JWT access token is propagated with the request.
- Token validation - Each service receiving the request validates the JWT access token by verifying its integrity, expiration, and issuer.
- Getting the identity - After a successful token validation, the service extracts the user's identity from the JWT access token's claims.
- Access Controls - Based on the identity, appropriate access controls are applied to determine the user's access to resources.

Oracle Integration provides support for OAuth identity propagation with the *OAuth using the JWT User Assertion* security policy. This security policy is available if you need to use the following adapters as invoke connections to call REST API operations.

- REST Adapter
- Oracle ERP Cloud Adapter
- Oracle HCM Cloud Adapter
- Oracle CX Sales and B2B Service Adapter

No identity propagation configuration tasks are required on the Connections page or in the Adapter Endpoint Configuration Wizard for these adapters. Instead, you configure the user identity to propagate in the mapper with the **Subject** element under **Security Properties**.

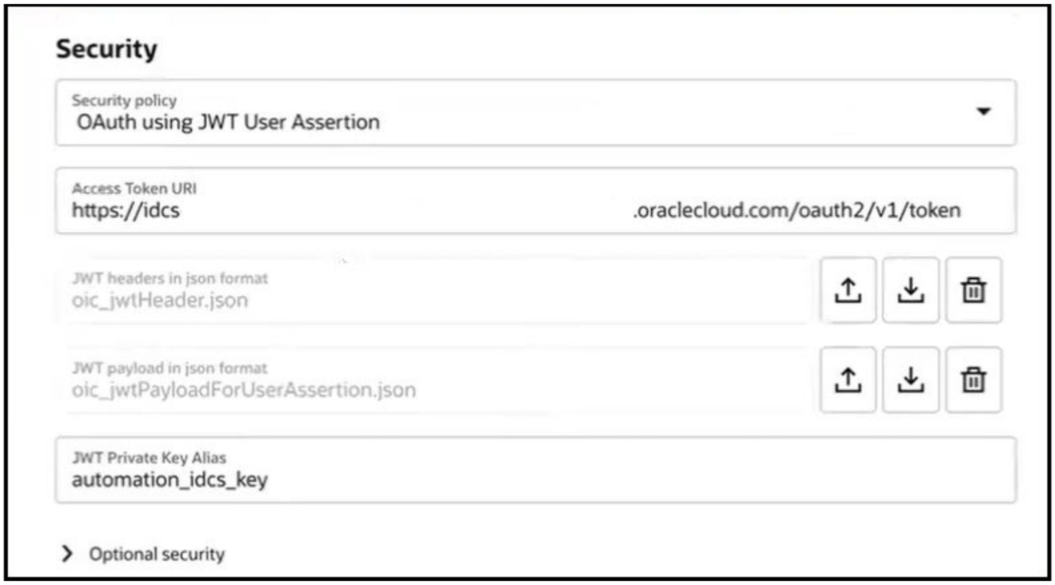
Note:

- To use identity propagation with an invoke connection created prior to Release 25.04, open the adapter in the Adapter Endpoint Configuration Wizard, click through each page, and click **Save**. These actions create the necessary **Subject** source and target elements in the mapper.
- Identity propagation is an optional feature. If you do not want to use identity propagation with the *OAuth using JWT User Assertion* security policy, leave the **Subject** elements empty.

Propagate User Identity

This section provides an overview of propagating user identity.

1. Create a trigger connection (for example, with the REST Adapter).
2. Create a new invoke connection with the REST Adapter.
3. Configure the invoke connection to use the *OAuth using JWT User Assertion* security policy, uploading the necessary header and payload files, specifying the private key alias uploaded on the Certificates page, and specifying the scopes under **Optional security**.



Security

Security policy: OAuth using JWT User Assertion

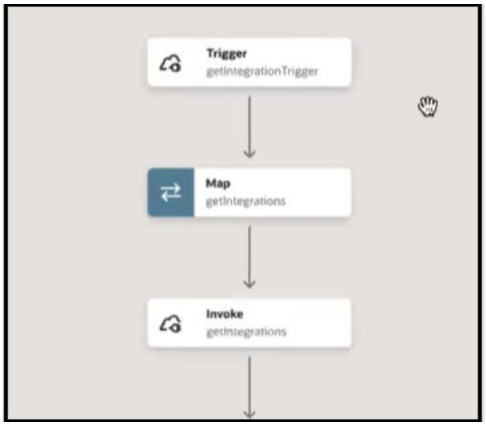
Access Token URI: https://idcs .oraclecloud.com/oauth2/v1/token

JWT headers in json format: oic_jwtHeader.json

JWT payload in json format: oic_jwtPayloadForUserAssertion.json

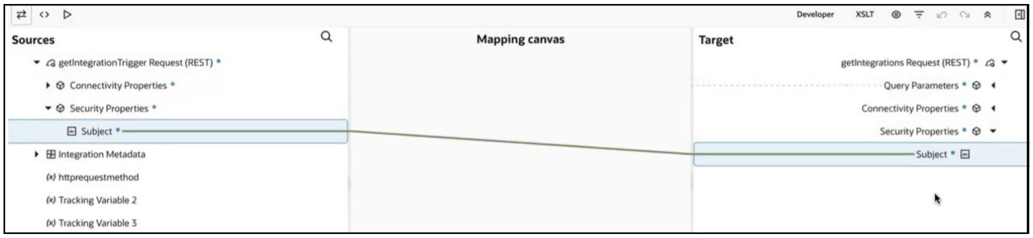
JWT Private Key Alias: automation_idcs_key

4. Create an application integration.
5. Drag the trigger and invoke connections into the integration canvas for configuration with the Adapter Endpoint Configuration Wizard.
6. Open the mapper.

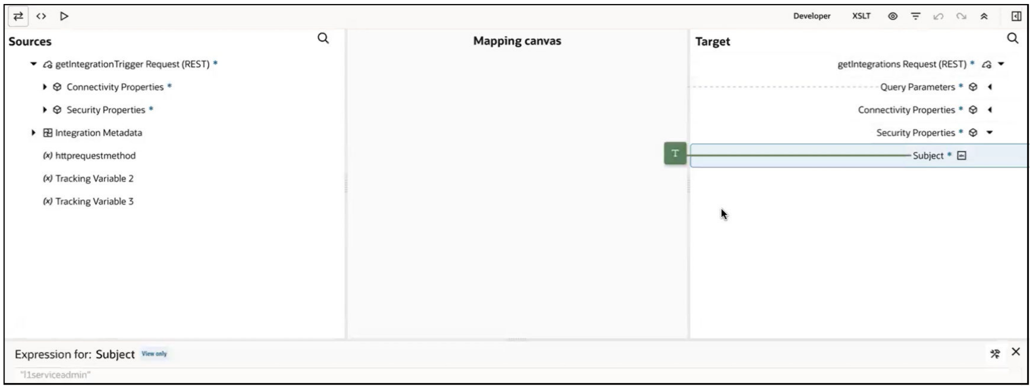


7. Set the user that has permission to execute the integration through either of two options in the mapper:

   - Expand **Security Properties** in the **Sources** and **Target** areas and map the user in the source **Subject** element to the target **Subject** element.

   - Expand **Security Properties** in the **Target** area and manually set the user for the **Subject** element in the Expression Builder. For this example, this option is demonstrated. The user name is specified in the Expression Builder as `"l1serviceadmin"`.

     Expression for: Subject
     "l1serviceadmin"

8. Complete design of your integration.

9. Activate and run the integration as the `l1serviceadmin` user specified in the mapper. The activity stream indicates the run was successful.

If you run the integration with a different user than that configured in the **Subject** element of the mapper, you receive the following error:

```
Request to access token failed. Cause: status = 400 Error:
{\"error\":\"invalid_grant\",\"error_description\":\"Invalid user
assertion:
The user name that you entered is invalid. Contact your system
administrator.\"
```

When you map the subject from trigger to invoke, the identity of the user who invokes the flow overrides the user configured in the JSON file. For example, at runtime, user1, user2, and user3 can run the integration flow and they'll all get tokens for each specific user. The JWT payload JSON file remains unchanged.

Watch a video to learn more:

Video

REST API Consumption Patterns

You can use the REST Adapter to implement the following common patterns to consume REST APIs.

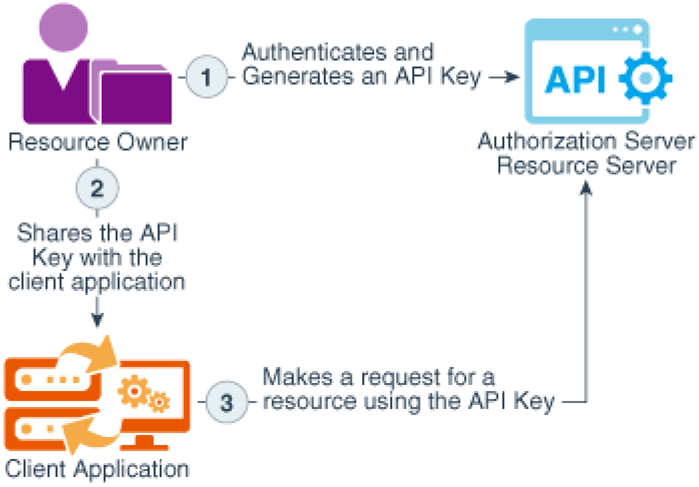
Topics:

- Configure the REST Adapter to Consume a REST API Protected with the API Key
- Configure the REST Adapter to Consume an External REST API with No Metadata Described in a Document
- Configure a REST Adapter to Consume a REST API that Expects Custom HTTP Header Properties
- Configure the REST Adapter to Consume an Amazon Web Services (AWS) REST API

Configure the REST Adapter to Consume a REST API Protected with the API Key

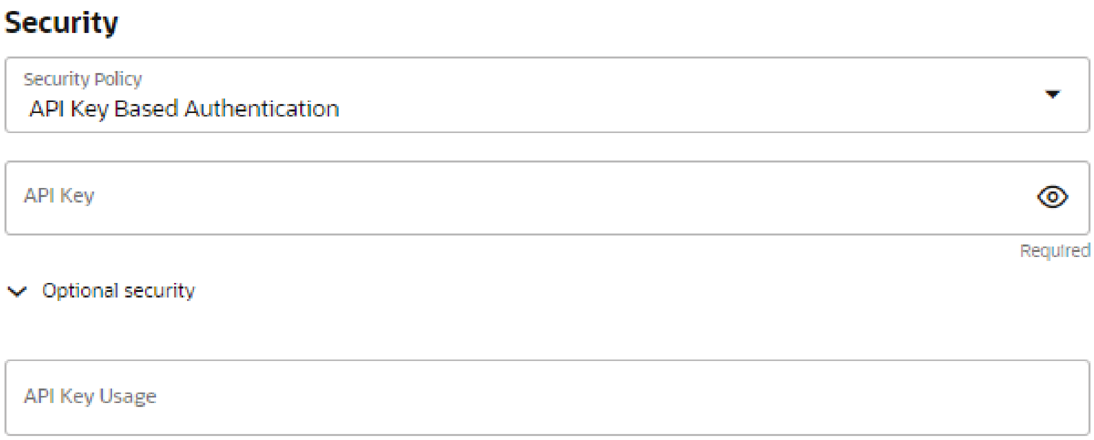
This section provides an overview of the API Key-Based Authentication security policy. This policy enables you to provide secure access to APIs. The resource owner generates an API key for a given client application with the required authorization and then shares the generated API key. The client application is then required to pass the API key with the request for accessing protected resources.

The following steps are performed as part of the API key-based authentication flow.


Resource Owner
1
Authenticates and Generates an API Key
API
Authorization Server
Resource Server
2
Shares the API Key with the client application
Client Application
3
Makes a request for a resource using the API Key

Step	Description
1	The resource owner authenticates and generates an API key for the given client application.
2	The resource owner shares the generated API key with the client application.
3	The client application makes a request for a resource using the API key.

On the Connections page of the REST Adapter, you select **API Key Based Authentication**.



Security

Security Policy: API Key Based Authentication

API Key

Required

Optional security

API Key Usage

In the **API Key Usage** field, you specify how the API key is passed with the request for accessing a resource. Enter this information carefully since this usage governs how the provided API key is passed to the endpoint. See Configure Connection Security for details.

At runtime, the API key is automatically passed to the endpoint while sending the request.

See Configure Connection Security.

Configure the REST Adapter to Consume an External REST API with No Metadata Described in a Document

Oracle Integration can integrate with REST APIs that do not publish any service description. The following example shows how to integrate with these REST APIs. This example uses a publicly available API that provides carbon intensity data for the United Kingdom.

> **Note:**
>
> The REST Adapter provides support for consuming REST APIs that are described using the metadata catalog. However, because the metadata catalog as a standard is not being actively maintained, you are advised to not use the metadata catalog definition. Many applications have already moved their resource models to the OpenAPI Specification, which is the preferred metadata description for describing RESTful APIs. If the metadata catalog happens to be the only metadata definition, you have the option of directly consuming the target REST API using the request builder provided out of the box as part of the Adapter Endpoint Configuration Wizard.

The API is described at `https://carbon-intensity.github.io/api-definitions/#intensity`. In this example, an integration is modeled to fetch carbon intensity data. Because the API is not protected, no security configuration is required.

The endpoint URL to invoke can also be invoked using the following `CURL` command:

```
curl -X GET https://api.carbonintensity.org.uk/intensity/date  -H 'Accept:
application/json'
```

The response is of the form:

```
{"data":[ {"from": "2018-01-20T12:00Z",
    "to": "2018-01-20T12:30Z",
    "intensity": {
      "forecast": 266,
      "actual": 263,
      "index": "moderate"
    }
}]
}
```

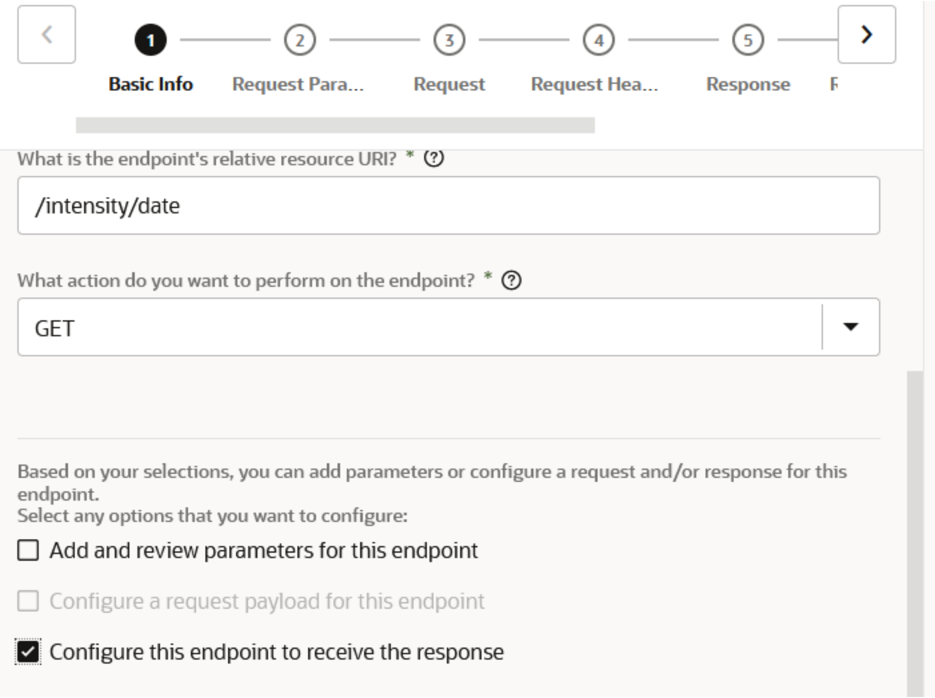
1. Configure a connection by selecting **REST API Base URL** in the **Connection Type** field and providing the base URL of the service in the **Connection URL** field. See Configure Connection Properties for Invoke Connections.

   Test and save the connection. Generally speaking, the REST API base URL should be the resource root of a REST API. In this example, the **Connection URL** field is configured as `https://api.carbonintensity.org.uk`.

   The following steps describe how to configure the relative REST resource in the Adapter Endpoint Configuration Wizard.

2. Configure the REST Adapter as an invoke connection. Oracle Integration determines the target endpoint URL by appending the relative resource URI to the base URL configured during connection configuration.
3. Provide a relative resource URI of `/intensity/date` and select the HTTP verb to use (**GET** for this example).

   In this example, a request payload is not required. Therefore, the corresponding option is not selected. The same applies for query and template parameters. However, since a response is expected, the option corresponding to a response is selected. The Adapter Endpoint Configuration Wizard determines the next page to show based on the options selected on this page.

   What is the endpoint's relative resource URI? *

   /intensity/date

   What action do you want to perform on the endpoint? *

   GET

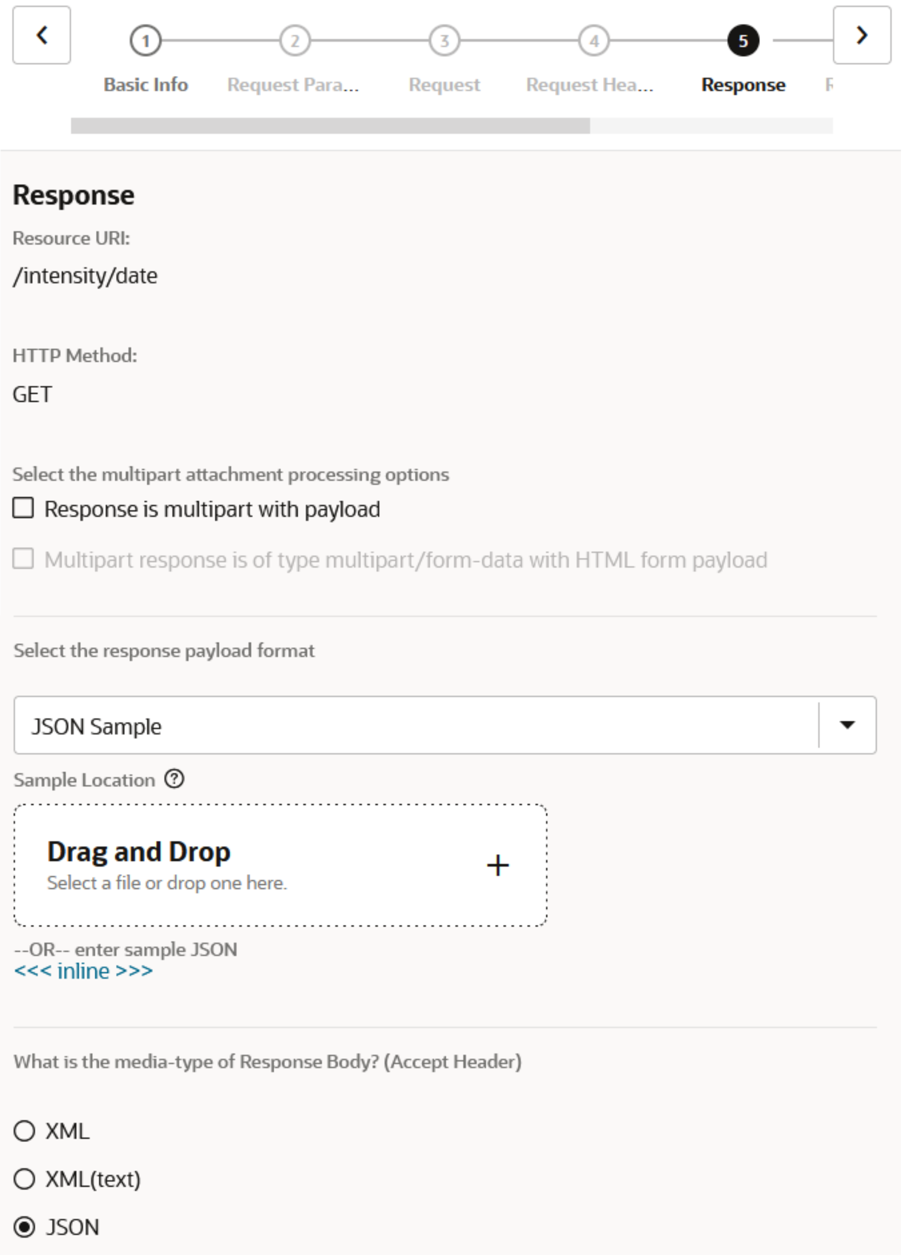
   Based on your selections, you can add parameters or configure a request and/or response for this endpoint.
   Select any options that you want to configure:

   ☐ Add and review parameters for this endpoint

   ☐ Configure a request payload for this endpoint

   ☑ Configure this endpoint to receive the response

   Because the options corresponding to request payload, request parameters (query and template parameters), and request headers were not selected, the corresponding pages are skipped.
4. Select the required payload format and provide a sample JSON, XML, or schema that represents the payload.



Response

Resource URI:

/intensity/date

HTTP Method:

GET

Select the multipart attachment processing options

□ Response is multipart with payload

□ Multipart response is of type multipart/form-data with HTML form payload

Select the response payload format

JSON Sample

Sample Location

Drag and Drop

Select a file or drop one here.

--OR-- enter sample JSON

<<< inline >>>

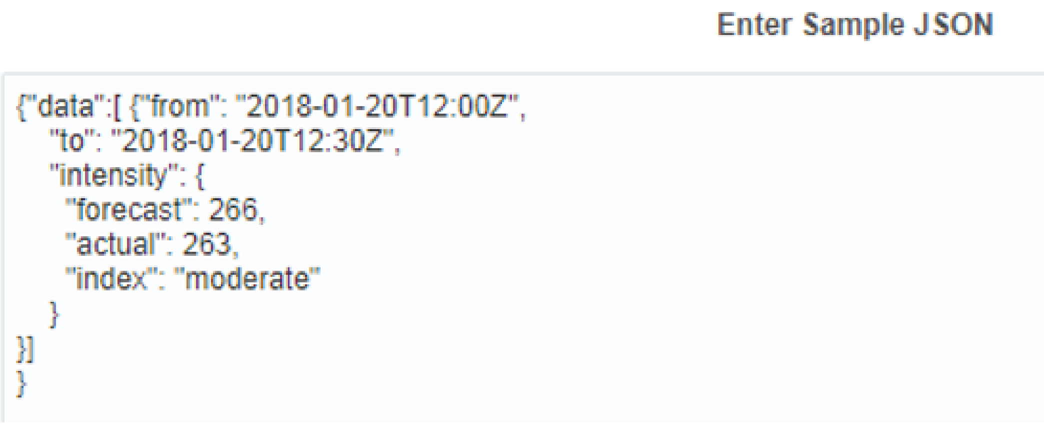
What is the media-type of Response Body? (Accept Header)

○ XML

○ XML(text)

● JSON

A JSON sample can also be provided using the **<<<inline>>>** option.



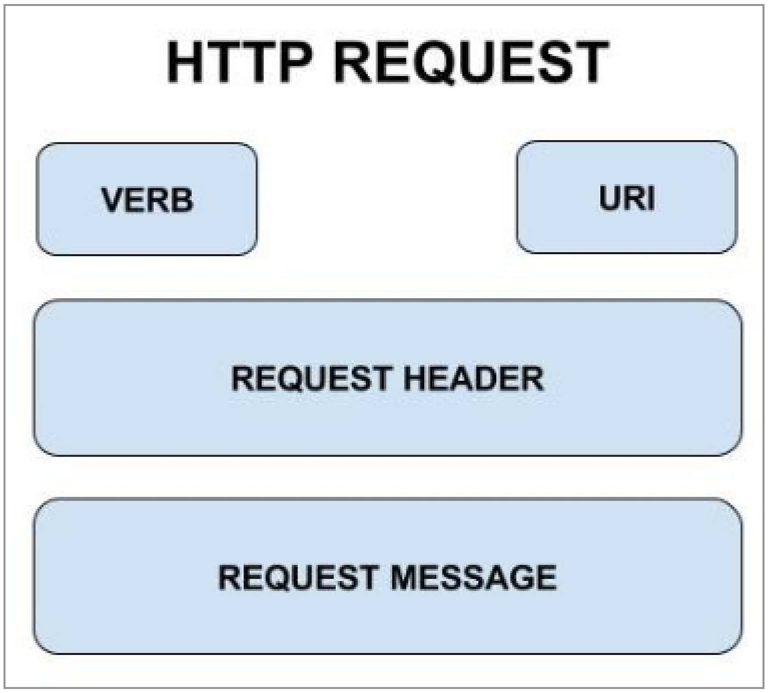
Enter Sample JSON

```
{"data":[ {"from": "2018-01-20T12:00Z",
   "to": "2018-01-20T12:30Z",
   "intensity": {
     "forecast": 266,
     "actual": 263,
     "index": "moderate"
   }
}]
}
```

5. Complete the rest of the Adapter Endpoint Configuration Wizard.
6. Complete the mappings.

Configure a REST Adapter to Consume a REST API that Expects Custom HTTP Header Properties

The REST Adapter provides an easy and configurable way to consume an external HTTP service. You can configure the HTTP verb, resource URI, query and template parameters, HTTP headers, form parameters, body, and attachments that must be sent as part of the request.



HTTP headers allow the client and the service to exchange additional information along with the request or the response. The Internet Assigned Numbers Authority (IANA) maintains a registry of standard or permanent HTTP request headers that are commonly used for predefined reasons. Along with the standard headers, services can also define custom proprietary headers for exchanging additional information.

Follow the steps mentioned below to invoke a REST service that expects a custom HTTP request header.

1. Create a connection with a REST Adapter invoke connection for the target service to consume.
2. Drag the connection onto the integration canvas.
3. On the Basic Info page, provide the HTTP verb and the relative request URI.
4. Select **Custom** in the **Configure Request Headers** section.

   The REST Adapter shows a page for you to configure the custom request headers.
5. Define the proprietary header name and provide a brief description of the header.

   Upon completion, the REST Adapter exposes the custom header specified above as part of the adapter request payload.
6. Assign this header a value using an assign action or the mapper. The assigned value is sent as a custom HTTP header to the target service at runtime.

Configure the REST Adapter to Consume an Amazon Web Services (AWS) REST API

You can configure the REST Adapter to consume an Amazon Web Services (AWS) REST API by selecting the AWS Signature Version 4 security policy on the Connections page. AWS provides a set of global compute, storage, database, analytics, application, and deployment services for consumption.

1. On the Connections page for the REST Adapter, go to the **Properties** section. At a minimum, specify the following details.
2. From the **Connection Type** list, select **REST API Base URL**.
3. In the **Connection URL** field, specify the endpoint URL according to the service you want to use. The REST Adapter exposes dynamic endpoint support. For example:

   ```
   https://amazonaws.com
   ```
4. Go to the **Security** section.
5. From the **Security Policy** list, select **AWS Signature Version 4**.
6. Specify the following details.

Element	Description
Access Key	Enter the access key obtained when you created your Amazon security credentials.
Secret Key	Enter the secret key obtained when you created your Amazon security credentials.
AWS Region	Select the region in which the AWS server is hosted.
Service Name	Select the AWS service to which to connect.

JSON Content Patterns

You can use the REST Adapter to implement the following common patterns with JSON content.

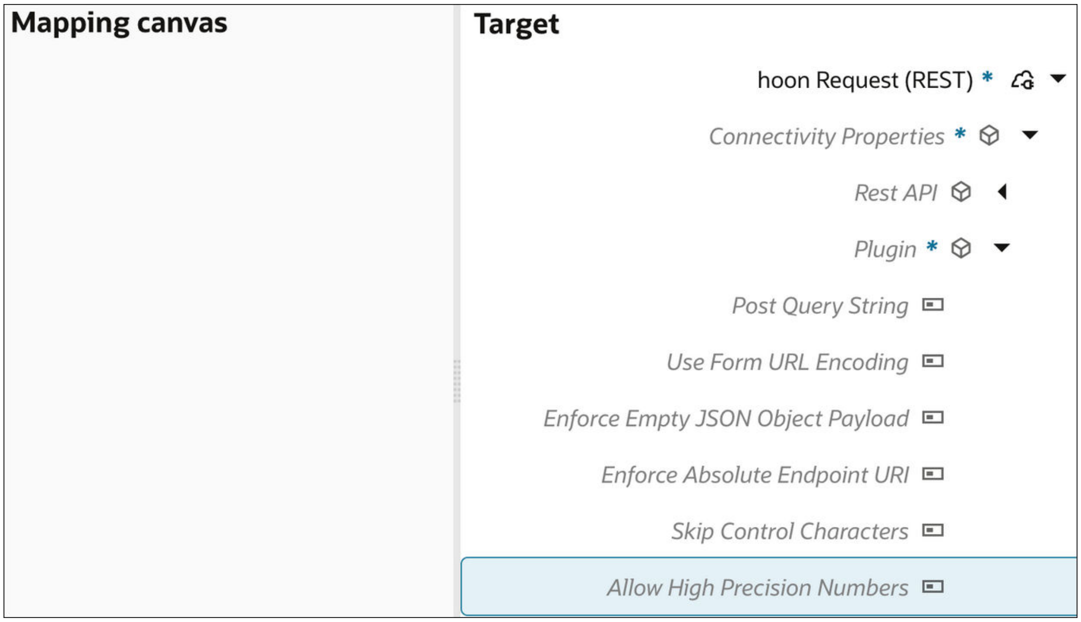
Topics:

- Allow JSON Numbers with High Precision and Scale
- Map JSON when the REST Adapter Request is Configured with multipart/form-data
- JSON to XML Special Character Conversion
- Send an Empty JSON Object
- Copy Element Names as Values in JSON
- Use JSON Objects With Single Elements Within an Array

Allow JSON Numbers with High Precision and Scale

For JSON numbers with high precision and scale, Oracle Integration automatically converts the number to four decimal places. To prevent this conversion, you have several options based on the adapter you are using.

- For the REST Adapter, you must set the **Allow High Precision Numbers** connectivity property in the mapper to allow JSON numbers with high precision and scale.
  The default value for this connectivity property is false. Oracle Integration internally restricts the precision scale to four (decimal places) by default. Set this property to true for a higher precision scale. Use this property value carefully because undefined precision does not mean infinite precision.



- For all other adapters that supply a schema for response processing, the following schema annotation is included.

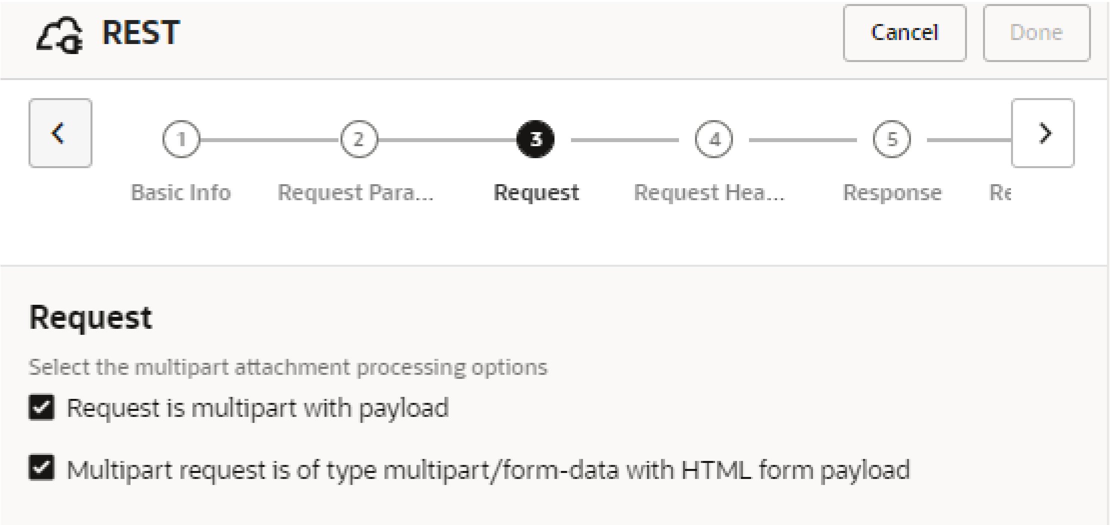
  ```
  <xs:schema xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
  nxsd:version="JSON" nxsd:allowHighPrecisionNumbers="true">
  ```

  If you receive a REST response with more than 10 decimal places, Oracle Integration converts it to XML, but does not round off the values.

  If you do not use this schema annotation, the value is rounded off at four decimal places.

Map JSON when the REST Adapter Request is Configured with multipart/form-data

JSON can be mapped when the REST Adapter request is configured with multipart/form-data (that is, when the **Request is multipart with payload** and **Multipart request is of type multipart/form-data with HTML form payload** check boxes are selected on the Request page).



Request

Select the multipart attachment processing options

☑ Request is multipart with payload

☑ Multipart request is of type multipart/form-data with HTML form payload

You can send a JSON string as a parameter. The name of the parameter is **jsonInputParameters**. The value of the parameter is the JSON string shown below. The value should be mapped to the **parameter** node. In general, **ParameterList** contains a list of parameters. Each parameter's name goes into **parameter** > **name** and its value goes into **parameter**.

JSON to XML Special Character Conversion

If the JSON payload has special characters that are not valid in XML, those characters are replaced by a string when converted from JSON to XML.

For example, assume you have the following JSON payload:`{ "_id": { "$oid": "52cdef7f4bab8bd67529c6f7" } }`

You then select the **JSON Sample** payload format and **<<inline>>** to copy and paste the payload into the text field in the Adapter Endpoint Configuration Wizard.

In the mapper, the field `$oid` is represented with a string value of `_0x646c72_oid`.

The list of special characters and their corresponding XML conversion strings are as follows:

Special Character	Converted Value Represented in the Mapper
`" "`	`_0x737063_`
`"/"`	`_0x736c68_`
`"\\"`	`_0x626c68_`
`":"`	`_0x636c6e_`
`";"`	`_0x73636e_`
`"("`	`_0x6c7072_`
`")"`	`_0x727072_`
`"&"`	`_0x616d70_`
`","`	`_0x636d61_`
`"#"`	`_0x706e64_`
`"?"`	`_0x717374_`
`"<"`	`_0x6c7374_`
`">"`	`_0x677274_`
`"start"`	`_0x737472_`
`"@"`	`_0x617472_`
`"$"`	`_0x646c72_`
`"{"`	`_0x6c6362_`
`"}"`	`_0x726362_`
`"%"`	`_0x706572_`

Send an Empty JSON Object

The REST Adapter can enforce the sending of an empty JSON object (`{}`) when needed with the **Enforce Empty JSON Object Payload** property. Some providers have this requirement in special cases, such as Concur. When enabled, this property inserts an empty JSON object.

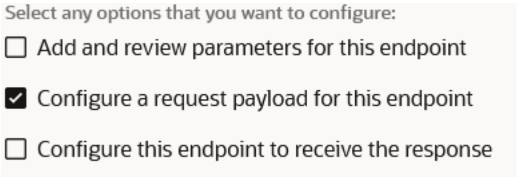
This property is driven by Concur and Oracle Logistics and is disabled by default.
When set to **true**, the REST call inserts `{}` into the body regardless of the verb chosen. For some third party APIs, this is a requirement for certain calls. By default, this property is not enabled.

The following steps provide a high level overview of how to configure an empty JSON object.

1. Add a REST Adapter as an invoke connection in the integration canvas.

   The Adapter Endpoint Configuration Wizard is displayed.

2. On the Basic Info page, select **Configure a request payload for this endpoint**.

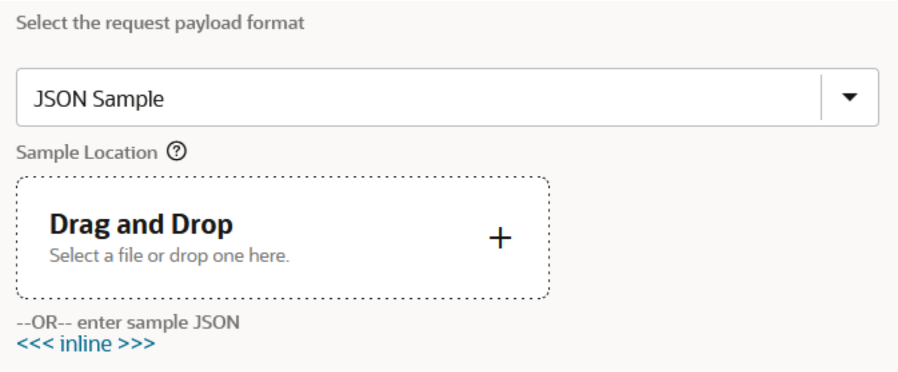
   Select any options that you want to configure:

   □ Add and review parameters for this endpoint

   ☑ Configure a request payload for this endpoint

   □ Configure this endpoint to receive the response

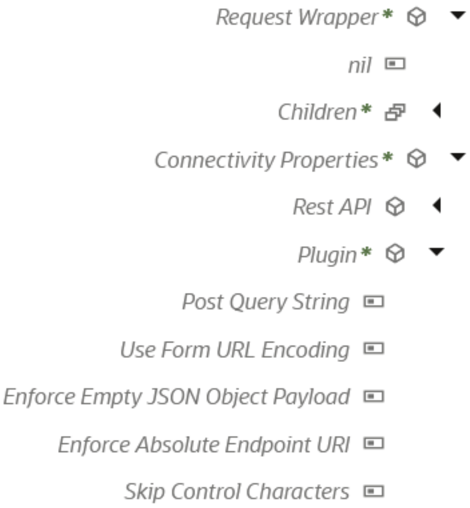
3. On the Request page, select **JSON Sample** from the **Select the request payload format** list, then specify `{}` as the empty JSON object in the **<<<inline>>>** link.



Select the request payload format: JSON Sample

4. Open the mapper in front of the REST Adapter.

5. Set the **Enforce Empty JSON Object Payload** target element to **true** in the mapper. Use the JavaScript function **true()**. You can enter this in the mapper without having to put quotes around around it. This creates a boolean value of **true**.



Copy Element Names as Values in JSON

You can copy element names as values in JSON.

JSON Sample	XSLT Mapping	Explanation

• Input
{
"OutputCollection":
{
    "Collection": [{
      "ID": 1,
      "NAME": "Name"
    },{
      "ID": 11,
      "NAME":
"Name0"
    }]
  }
}
• Output
{
  "items" : [{
    "name" : "ID",
    "value" :
["1"]
  }, {
    "name" : "NAME",
    "value" :
["Name"]
  }, {
    "name" : "ID",
    "value" :
["11"]
  }, {
    "name" : "NAME",
    "value" :
["Name0"]
  }]
}
	<xsl:for-each select="/
	nstrgmpr:execute/nstrgdfl:request-
	wrapper/nstrgdfl:OutputCollection/
	nstrgdfl:Collection/*">
	  <nstrgdfl:items>
	    <nstrgdfl:name>
	      <xsl:value-of select="name
	()"/>
	    </nstrgdfl:name>
	    <nstrgdfl:value>
	      <xsl:value-of select="current
	()"/>
	    </nstrgdfl:value>
	  </nstrgdfl:items>
	</xsl:for-each>
		Iterate over child elements of a required element. Capture the name and value as required.

Use JSON Objects With Single Elements Within an Array

You can use JSON objects with single elements within an array.

JSON Sample	XSLT Mapping	Explanation

`{`
`  "parameters": [`
`    {`
`      "closeAction": "closeAction"`
`    },`
`    {`
`      "closeReason": "closeReason"`
`    }`
`  ]`
`}`
	`<execute>`
	`  <parameters>`
	`    <closeAction>closeAction</closeAction>`
	`  </parameters>`
	`  <parameters>`
	`    <closeReason>closeReason</closeReason>`
	`  </parameters>`
	`</execute>`
		An array in JSON is a repeating element in XML. An XML element is an item in a JSON object.

OpenAPI Document Patterns

You can use the REST Adapter to implement the following common patterns using OpenAPI documents.

Topics:

- Publish REST-Based Integrations as OpenAPI Documents
- Consume and Publish OpenAPI Documents with Multipart/Mixed and Multipart/Form-Data

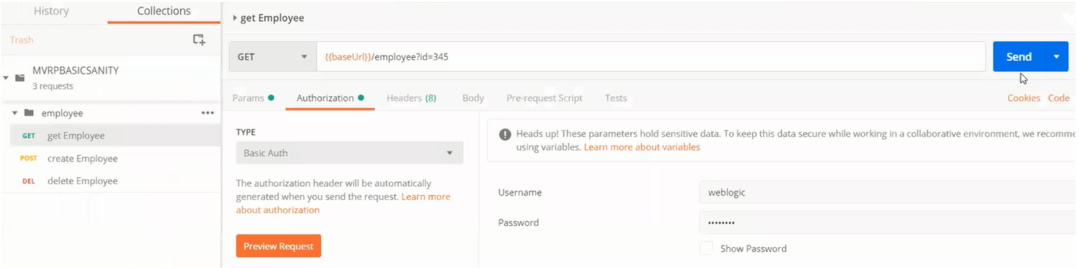
Publish REST-Based Integrations as OpenAPI Documents

You can publish an OpenAPI document describing an Oracle Integration REST endpoint and consume the OpenAPI document using the REST Adapter. This section provides an overview of the configuration steps.

1. Create a REST Adapter connection. The connection can use any security policy.
2. On the Integrations page, create an application integration.
3. Add the REST Adapter created in Step 1 as the trigger connection.
4. Design the integration.
5. On the Integrations page, activate the integration.
6. Click ..., then **Run** to open the Run and configure page for running the integration.
7. Click the **Endpoint Metadata** value.

   This shows a message with a link to the Endpoint URL, Swagger, and Open API descriptions of the REST endpoint.
8. Click **Open API** to display the description for this endpoint, including the server to invoke, the exposed resources, and security details.

9. To invoke the integration, import the document into a client such as postman.
10. Select an resource to invoke, and click **Send**. For this example, **getEmployee** is selected. This matches the selection made on the Operation Selection page of the Adapter Endpoint Configuration Wizard.



GET {{baseUrl}}/employee?id=345

TYPE: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recomme using variables. Learn more about variables

Username: weblogic

Password: ••••••••

□ Show Password

Consume and Publish OpenAPI Documents with Multipart/Mixed and Multipart/Form-Data

You can consume and publish OpenAPI documents with multipart/mixed and multipart/form-data in REST Adapter trigger connections. This section describes how to configure this type of scenario in Oracle Integration.

- Consume an OpenAPI Document with Endpoints that Consume or Produce HTML Form Data
- Consume an OpenAPI Document with Endpoints with a Multipart/mixed Content Type
- Consume an OpenAPI Document with Endpoints with multipart/form-data

Consume an OpenAPI Document with Endpoints that Consume or Produce HTML Form Data

1. Create a REST Adapter connection and select the **Trigger and Invoke** role.
2. In the **Connection Properties** section, select **OpenAPI 1.0/2.0/3.0 URL** and specify the connection URL.
3. Create an application integration.
4. Add a REST Adapter trigger connection.
5. On the Resource Configuration page, select a **POST** action with **Configure a request payload for this endpoint** and **Configure this endpoint to receive the response**.
6. On the Request page, select **Request is multipart with payload** and **Multipart request is of type multipart/form-data with HTML form payload**.
7. On the Response page, select **Request is multipart with payload** and **Multipart request is of type multipart/form-data with HTML form payload**.
8. Add a second REST Adapter as an invoke connection to call the OpenAPI endpoint.
9. On the Operation Selection page, select **/formdata_html** from the **Resource** list to call the endpoint.
10. In the request mapper between the two REST Adapter connections, map the data sent to the trigger connection to the outbound request in the invoke connection. The **Parameter List** > **Parameter** > **Name** element for both the source and target includes the HTML form data.

11. In the response mapper after the second REST Adapter connection, perform the same mappings for the response from the outbound call.
12. Configure the business identifier to track the integration during runtime.
13. Save and activate the integration.
14. Invoke the integration and view the results.

Consume an OpenAPI Document with Endpoints with a Multipart/mixed Content Type

1. Create a REST Adapter connection and select the **Trigger and Invoke** role.
2. In the **Connection Properties** section, select **OpenAPI 1.0/2.0/3.0 URL** and specify the connection URL.
3. Create an application integration.
4. Add a REST Adapter trigger connection.
5. On the Resource Configuration page, select a **POST** action with **Configure a request payload for this endpoint** and **Configure this endpoint to receive the response**.
6. On the Request page, select the following:
   a. Select **Request is multipart with payload**.
   b. Select **JSON Sample** in the dropdown and supply a valid JSON sample object.
   c. Select **multipart/mixed**.
7. On the Response page, select the following:
   a. Select **Request is multipart with payload**.
   b. Select **JSON Sample** in the dropdown and supply a valid JSON sample object.
   c. Select **multipart/mixed**.
8. Add a second REST Adapter as an invoke connection to call the OpenAPI endpoint.
9. On the Operation selection page, select **/mixed_json** from the **Resource** list to call the endpoint.
10. In the request mapper between the two REST Adapter connections, map the data sent to the trigger connection to the outbound request in the invoke connection. **Contact Disposition** is set to a concrete value of **"form-data"**. It must be form data for multipart mixed to work. The **Mixed Json Post Request and Multipart Mixed Data** target node represents the payload.
11. In the response mapper after the second REST Adapter connection, perform the same mappings for the response from the outbound call.
12. Configure the business identifier to track the integration during runtime.
13. Save and activate the integration.
14. Invoke the integration and view the results.

Consume an OpenAPI Document with Endpoints with multipart/form-data

1. Create a REST Adapter connection and select the **Trigger and Invoke** role.
2. In the **Connection Properties** section, select **OpenAPI 1.0/2.0/3.0 URL** and specify the connection URL.
3. Create an application integration.
4. Add a REST Adapter trigger connection.

5. On the Resource Configuration page, select a **POST** action with **Configure a request payload for this endpoint** and **Configure this endpoint to receive the response**.
6. On the Request page, select the following:
   a. Select **Request is multipart with payload**.
   b. Select **JSON Sample** in the dropdown and supply a valid JSON sample object.
   c. Select **multipart/form-data**.
7. On the Response page, select the following:
   a. Select **Request is multipart with payload**.
   b. Select **JSON Sample** in the dropdown and supply a valid JSON sample object.
   c. Select **multipart/form-data**.
8. Add a second REST Adapter as an invoke connection to call the OpenAPI endpoint.
9. On the Operation selection page, select **/formdata_json** from the **Resource** list to call the endpoint.
10. In the request mapper between the two REST Adapter connections, map the data sent to the trigger connection to the outbound request in the invoke connection. The **Formdata Json Post Request Multipart Form Data Data** target node represents the payload.
11. In the response mapper after the second REST Adapter connection, perform the same mappings for the response from the outbound call.
12. Configure the business identifier to track the integration during runtime.
13. Save and activate the integration.
14. Invoke the integration and view the results.

Best Practices for Invoking REST Endpoints

Follow these best practices for invoking REST endpoints with the REST AdapterREST Adapter.

- If you receive errors (for example, `401`, `429`, or `50x` errors) while invoking REST endpoints with the REST Adapter, ensure that you employ instance retries.
- Client applications should cache the token while invoking OAuth-protected REST endpoints.

Override the Endpoint URI/Host Name for an External REST API at Runtime

You can design integrations in Oracle Integration in which you specify an endpoint URI at runtime to invoke an external REST API. This feature is useful in situations in which the endpoint of the external REST API is either not known at design time or a decision must be made by the integration at run time to determine which one of the multiple REST services must be invoked.

Perform the following steps to configure an integration to invoke a REST endpoint dynamically using Oracle Integration.

The integration is typically designed with the REST Adapter as an invoke connection. The connection has either the base URI or the absolute endpoint URI specified in a Swagger document. In either case, the endpoint URI for the external API is derived at design time, and is static.

In scenarios in which the endpoints are overridden at run time, it is assumed that the APIs hosted on these endpoints comply with the interface defined for the API at design time.

1. Create and configure a REST Adapter as an invoke connection.

   During design time configuration, the interface for the external API is being specified declaratively: the shape of the request and the response message (if any), the HTTP method used, and the message exchange pattern (request, response, or one way).
2. In the **Target** section of the mapper, expand **RestApi** under **ConnectivityProperties**.
3. From the **Source** schema, provide a mapping for **AbsoluteEndpointUri**. **AbsoluteEndpointUri** must be assigned the endpoint URI that has concrete values for the path/template parameters and any query parameters with values. The REST Adapter sends the request to the address stored in this property. Alternatively, you can also provide a static mapping.
4. Activate and invoke the integration. The REST Adapter uses the runtime value provided by this mapping to determine the REST endpoint to which to route this request.
5. Alternatively, in Step 4, you can map other siblings of **AbsoluteEndpointUri**. For a finer control, you can also provide mappings for individual components of the URI by expanding the URI.
   - **Scheme**: Provide a mapping if you want to change only the scheme of the endpoint URL. Supported values are **HTTP** and **HTTPS** only.
   - **Host**: Provide a mapping if you want to change only the host portion of the endpoint URL.
   - **Port**: Provide a mapping if you want to change only the port of the endpoint URL.
   - **Query**: Provide a mapping if you want to change only the query portion of the endpoint URL. A query portion is the one that follows the **?** character.
   - **Path**: Provide a mapping if you want to change only the path portion of the endpoint URL. A path is the part of a URI between the hostname and the **?** character.

Map to Construct the Payload for an External REST API that Accepts multipart/form-data as the Content Type

This section describes the data structures for different types of configurations made using the REST Adapter or any application adapter exposing the REST API (used as a trigger connection) or consuming the REST API (used as an invoke connection).

Categories

There are two categories of multipart request (and response):

- multipart/mixed or multipart/form-data configured with a JSON or XML sample

  This category shows the attachments schema and payload schema. The payload schema is derived based on the sample JSON/XML schema provided during Adapter Endpoint Configuration Wizard configuration.
- multipart/form-data with HTML form payload

  This is used when you select **Multipart request is of type multipart/form-data with HTML form payload** on the Request page of the Adapter Endpoint Configuration Wizard or when you select **Multipart request is of type multipart/form-data with HTML form payload** on the Response page for a response. This configuration shows the attachments schema and a generic schema with a **ParameterList** element. The **ParameterList** element consists of an unbounded element parameter. Each parameter has a **name** attribute. The value of the parameter is set directly to the **parameter** element. If there are multiple parameters, the **parameter** element can be repeated in the mapper.

Attachments Schema

The **attachments** element has an unbounded **attachment** element. This provides support for receiving (on the source) or sending (on the target) multiple attachments. Each **attachment** element has **attachmentReference** and **attachmentProperties**.

The **AttachmentReference** element contains the location where the attachment has been staged for access.

The **AttachmentProperties** element provide metadata about a single attachment. The **attachmentProperties** element is used follows:

- The **contentId** property is used to set the **Content-ID** header of the body part. The **Content-ID** header is used to set a unique ID for the body part.
- The **contentType** property is used to set the **Content-Type** header of the body part. For example, if a PDF file is being sent, the **contentType** property should be **application/pdf**. If the source is providing a multipart attachment, this is determined automatically. The mapper can set/override these values.
- The **transferEncoding** property is used to set the **Content-Transfer-Encoding** header of the body part. This header's value is a single token specifying the type of encoding, as enumerated below. Formally:

  ```
  Content-Transfer-Encoding := "BASE64" / "QUOTED-PRINTABLE" /
                               "8BIT"   / "7BIT" /
                               "BINARY" / x-token
  ```

  These values are not case sensitive. That is, **Base64**, **BASE64**, and **bAsE64** are all equivalent. An encoding type of **7BIT** requires that the body is already in a seven-bit, mail-ready representation. This is the default value; that is, **Content-Transfer-Encoding: 7BIT** is assumed if the **Content-Transfer-Encoding** header field is not present. See https://www.w3.org/Protocols/rfc1341/5_Content-Transfer-Encoding.html.
- The **partName** property is used to set the **fileName** of the body part. The attached **file/bodypart** should be saved by a target system with this name.
- The **contentDisposition** property is used to set the **Content-Disposition** of the body part.

  In a multipart/form-data body, the HTTP **Content-Disposition** is a header used on the subpart (that is, attachment) of a multipart body to provide information about the field to which it applies. The **Content-Disposition** header value is typically set to **form-data**. The optional directive name and filename can be used. For example:

  ```
  Content-Disposition: form-data
  Content-Disposition: form-data; name="fieldName"
  Content-Disposition: form-data; name="fieldName"; filename="filename.jpg"
  ```

  and so on.
- The **contentDescription** property is used to set some descriptive information with a given body part. For example, it may be useful to mark an **image** body as **a picture of the**

Space Shuttle Endeavor. Such text may be placed in the **Content-Description** header field.

- The **fileInputHtmlFieldName** property lets you set the name of the part from which the server needs to read the file. This is generally used when there is an HTML form to upload the file. The file upload input field name is used as a body part name.

Scenario 2 - source is multipart/mixed or multipart/form-data with JSON/XML payload (Category A). Outbound request multipart/form-data with form fields (Category B)

The following map focuses on mapping the attributes in the HTML form. There should be as many parameters in the **parameterList** as there are fields in the HTML form.

> **Note:**
>
> If the source is not multipart and the target must be multipart/form-data with form fields, you must specify the value for the **contentType** and **partName** elements on the target side.

The **fileInputHtmlFieldName** element is important to consider if the target endpoint expects attachments under a specific body part name. The body part name should be specified here.

Scenario 3 - creating a reference from base64–encoded content

In this scenario, the source has a base64-encoded string and the target can be either one of the three: multipart/mixed or multipart/form-data with JSON/XML payload, or multipart/form-data with HTML form payload.

In the inbound payload, the **content** element is a base64-encoded string. This can be sent as an attachment in the outbound request. The base64 string can be converted into a reference using XSL function **decodeBase64ToReference** and the reference can be assigned to the **attachmentReference** element as shown below. Since the inbound request is not multipart, but the outbound must be multipart, you must set some multipart-specific properties in the mapper for outbound. The **contentType** is set to **image/png**, **partName** is set to **picture.png**, and **fileInputHtmlFieldName** is set to **image**. The assumption is that the target system is configured to read from a body part having **name="image"** in its content disposition. This is done with the **fileInputHtmlFieldName** element.

> **Note:**
>
> If the target is multipart/mixed or multipart/form-data using a JSON/XML payload, the schema of the target also has the schema from JSON/XML, as shown in Scenario 1. The approach for constructing the outbound request payload is the same.

Implement an Integration in which to Send an Incoming Message with a Base64-Encoded String to an External REST API that Accepts a Multipart Attachment

In the inbound payload, the **content** element is a Base64–encoded string. This can be sent as an attachment in an outbound request.

The Base64 string can be converted into a reference using XSL function **decodeBase64ToReference**. The reference can be assigned to an **attachmentReference** element as described in this section.

Since the inbound request is not multipart, but the outbound must be multipart, you must set some multipart-specific properties in the mapper for the outbound direction.

In the mapper, the **contentType** element is set to **image/png**, **partName** is set to **picture.png**, and **fileInputHtmlFieldName** is set to **image**.

In this scenario, the assumption is that the target system is configured to read from a body part having **name="image"** in its content disposition. This is done through the **fileInputHtmlFieldName** element.

> **Note:**
>
> If the target is multipart/mixed or multipart/form-data using a JSON/XML payload, the schema of the target also has the schema from JSON/XML. The approach for constructing the outbound request payload is the same.

Pass the Payload as URL-Encoded Form Data

You can pass the payload as URL-encoded form data with the REST Adapter.

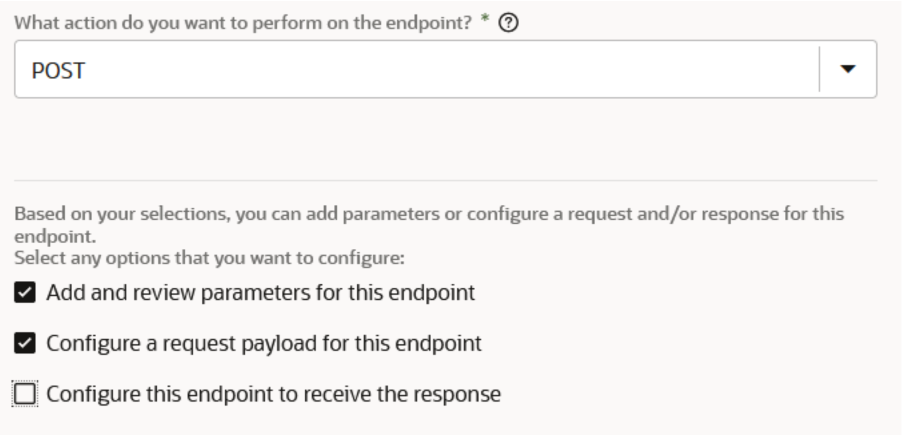
1. Add a REST Adapter as an invoke connection in an integration.

   The Adapter Endpoint Configuration Wizard is displayed.

2. On the Basic Info page, make the following selections:

   a. Select the **POST** verb.

   b. Select the **Add and review parameters for this endpoint** and **Configure a request payload for this endpoint** options.



What action do you want to perform on the endpoint? * ?

POST

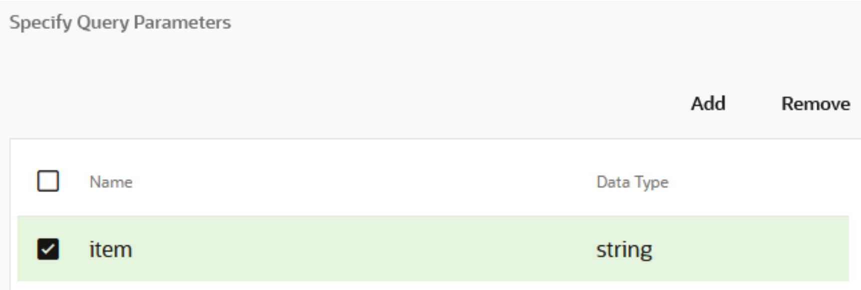
Based on your selections, you can add parameters or configure a request and/or response for this endpoint.
Select any options that you want to configure:

☑ Add and review parameters for this endpoint

☑ Configure a request payload for this endpoint

☐ Configure this endpoint to receive the response

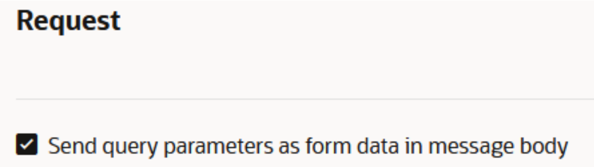
3. On the Request Parameters page, specify the query parameters and values to use.



Specify Query Parameters

□	Name	Data Type
☑	item	string

4. On the Request page, scroll to the bottom and select the **Send query parameter as form data in message body** checkbox. This enables the query parameters you defined on the Request Parameters page to be sent as URL-encoded form data in the payload.



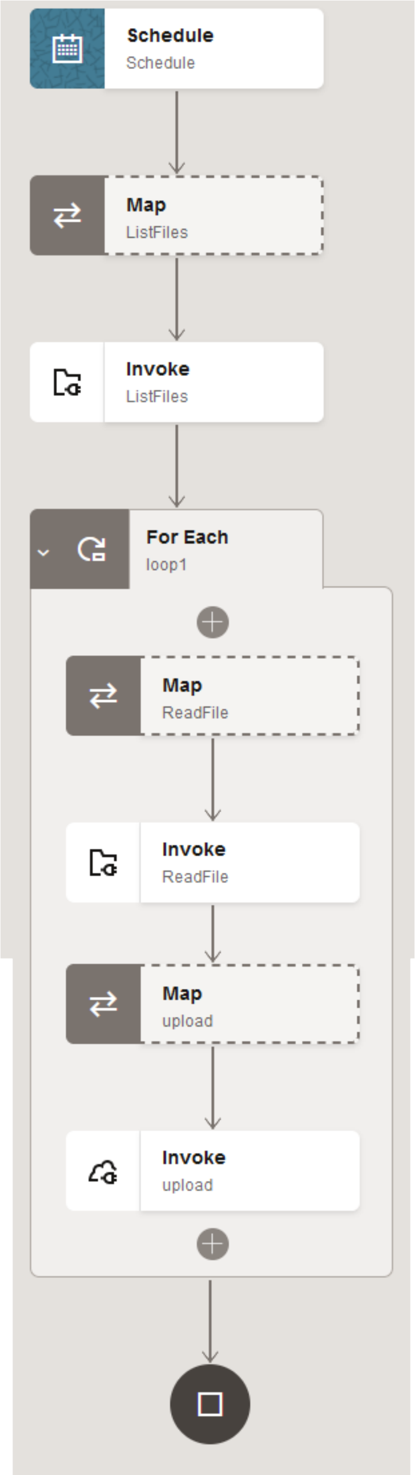
Request

☑ Send query parameters as form data in message body

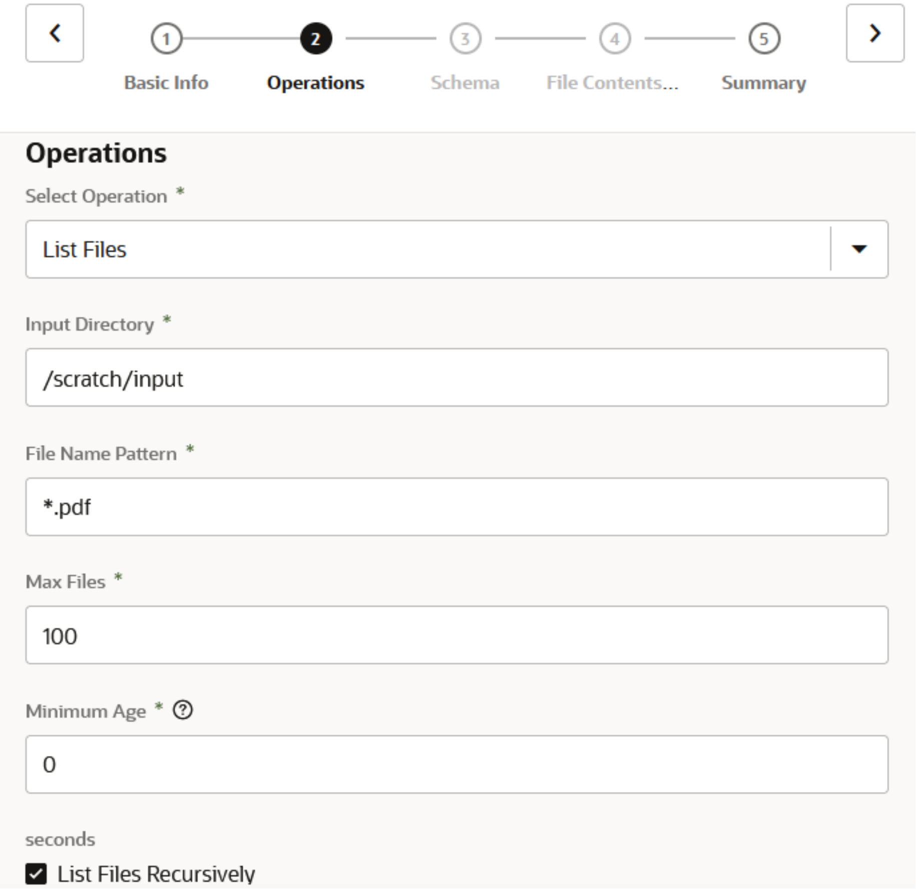
Implement an Integration to Send a PDF/CSV Document Downloaded from an SFTP Server to an External REST API that Accepts Only application/octet-stream as the Content Type

In an integration, the FTP adapter is only supported as an invoke connection. Implement this use case by selecting either an **Application** or **Schedule** integration on the Integration Style dialog.

The following example provides a high level overview on how to implement this use case as a schedule integration.



1. Configure an FTP Adapter with the **List Files** operation to list files from an SFTP server.



Operations

Select Operation *

List Files

Input Directory *

/scratch/input

File Name Pattern *

*.pdf

Max Files *

100

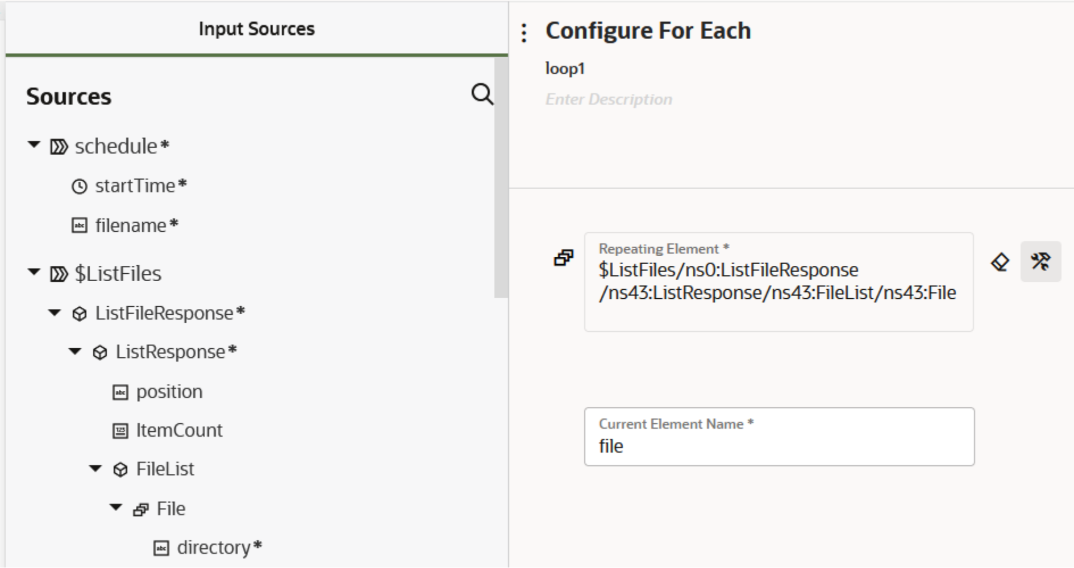
Minimum Age *

0

seconds

☑ List Files Recursively

2. Configure a for-each action to iterate through the **List Files** operation response.



Configure For Each

loop1

Enter Description

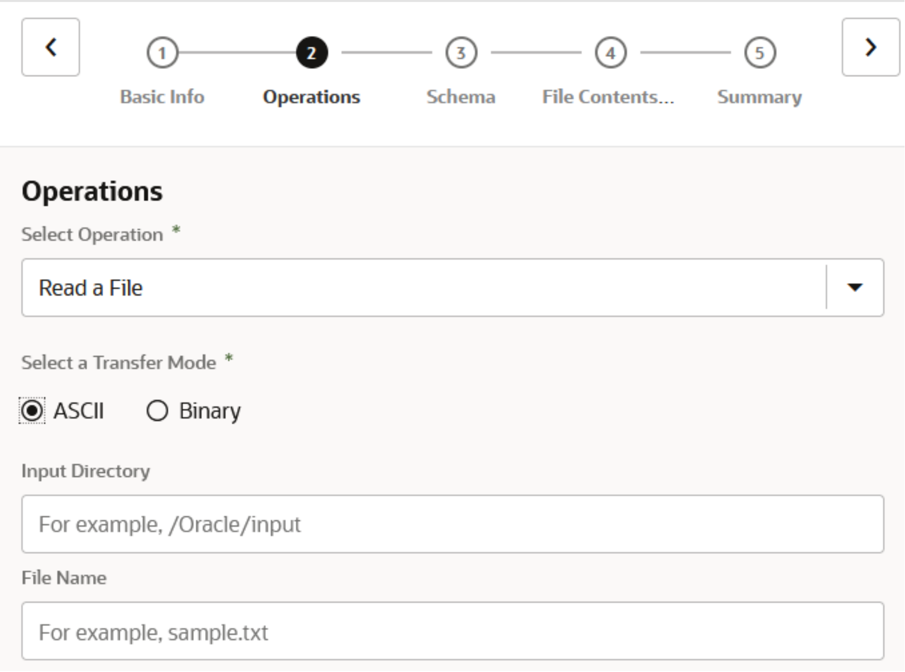
Repeating Element *

$ListFiles/ns0:ListFileResponse
/ns43:ListResponse/ns43:FileList/ns43:File

Current Element Name *

file

3. Configure a second FTP Adapter with the **Read Files** operation to read individual files inside the loop.



Operations

Select Operation *

Read a File

Select a Transfer Mode *

● ASCII ○ Binary

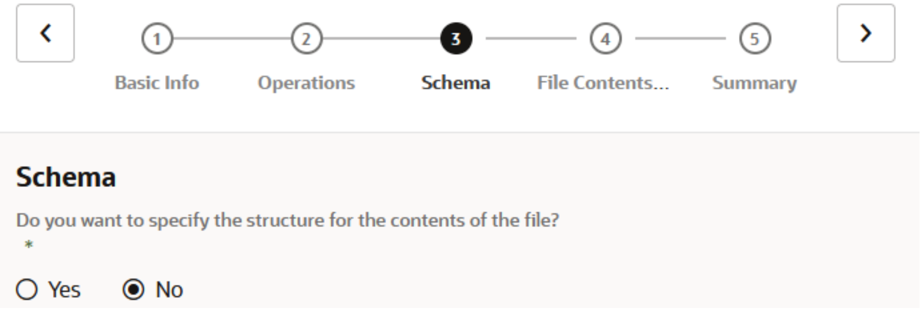
Input Directory

For example, /Oracle/input

File Name

For example, sample.txt

4. Configure the FTP Adapter to not specify a structure of the file.

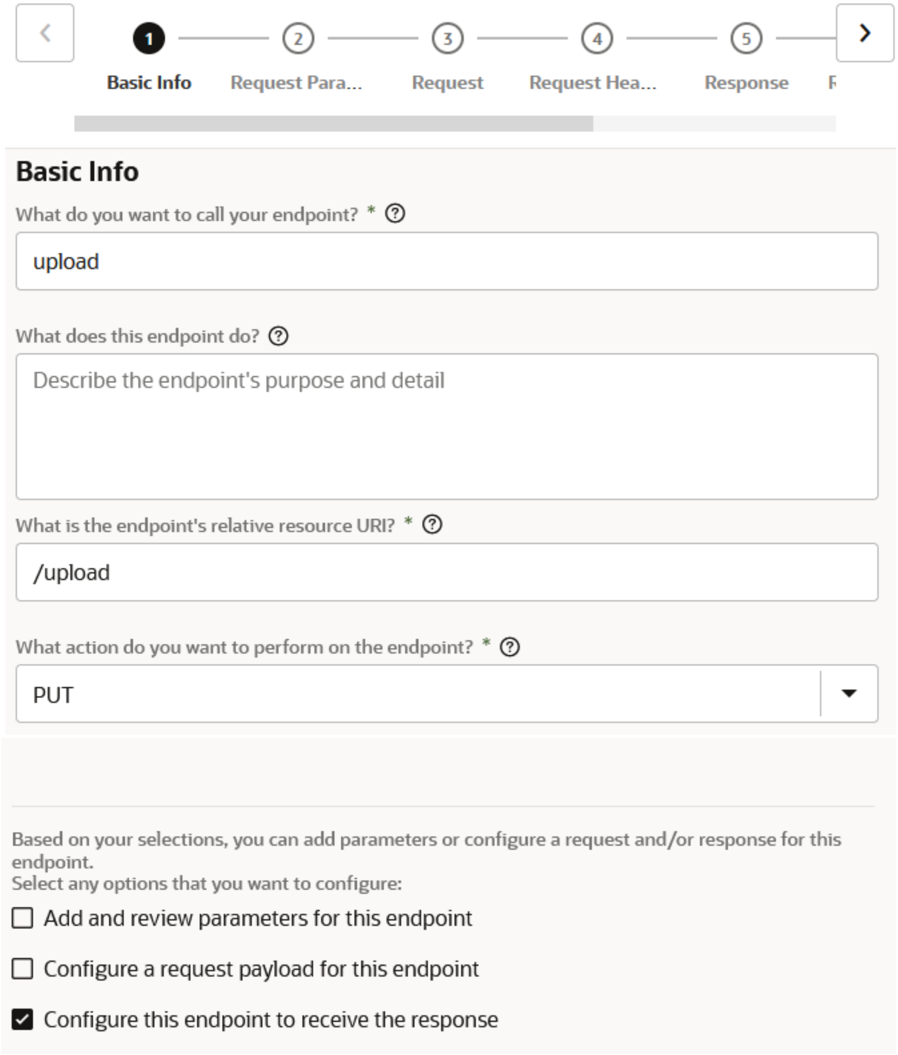


Schema

Do you want to specify the structure for the contents of the file? *

○ Yes ● No

5. Configure a REST Adapter. The reference of the **Read Files** operation is handed over to the outbound REST Adapter.



Basic Info

What do you want to call your endpoint? *

upload

What does this endpoint do?

Describe the endpoint's purpose and detail

What is the endpoint's relative resource URI? *

/upload

What action do you want to perform on the endpoint? *

PUT

Based on your selections, you can add parameters or configure a request and/or response for this endpoint.
Select any options that you want to configure:

☐ Add and review parameters for this endpoint

☐ Configure a request payload for this endpoint

☑ Configure this endpoint to receive the response

6. Configure the REST Adapter payload as **application/octet-stream**.
7. Configure the mapper to read individual files in the for-each loop.
   a. Perform the following **Source** to **Target** mappings:
      - **$file** > **File** > **directory** to **SyncReadFile** > **FileReadRequest** > **directory**
      - **$file** > **File** > **filename** to **SyncReadFile** > **FileReadRequest** > **filename**
8. Configure the mapper to send the read file to the outbound REST Adapter.
   a. Perform the following **Source** to **Target** mapping:
      - **$ReadFile** > **SyncReadFileResponse** > **FileReadResponse** > **ICSFile** > **FileReference** > to **execute** > **streamReference**.

Configure the REST Adapter to Expose an Integration as a REST API

Oracle Integration provides an easy way to expose an integration as a RESTFul service by using the REST Adapter as a trigger.

1. Create and test a REST Adapter connection with a trigger as the role.
2. Create an integration.
3. Drag the REST Adapter connection as a trigger within the integration canvas.
4. Follow the Adapter Endpoint Configuration Wizard to describe the shape of the RESTful service.

   The REST Adapter can be set up to accept a request on a specific URL path. This path can have template and query parameters. You can also decide on the HTTP method, the structure of the request payload, the request headers, and the CORS configuration. Likewise, you can also specify the response payload and the response headers that must be sent back to the client.
5. Upon activation, a RESTful service protected using OAuth and HTTP Basic Auth is created.

   A Swagger 2.0–compliant document is automatically produced for REST APIs exposed using the REST Adapter. This document describes the metadata for the generated REST APIs. Human-readable HTTP metadata is also produced for the REST endpoint.

Enter q as a Standard HTTP Query Parameter with the Query as a Value

Many APIs have special handling for the `q` query parameter according to different schemes, such as mongoDB query/SCIM/open search, and so on.

The REST Adapter treats `q` as a standard HTTP query parameter and treats the query expression as a string value. For example:

- `https://`*`host`*`.example.com:7004/resource?q=AssetNumber=AP10001`
- `GET /ccadmin/v1/products?q=orderLimit lt &maxLimit and startTime gt &startTime`
- `https://mysite.example.com/services/rest/connect/v1.3/queryResults?query=SELECT Contact from contract where contact.organization.name=&OrgName;`

According to standard HTTP, the query parameter in this case is `q` and the value is `AssetNumber=AP1000`.

Therefore, you are required to pass the query expression as a value to the query parameter with the name `q`.

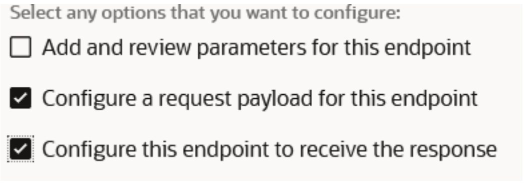
Configure Oracle Integration to Call Oracle Cloud Infrastructure Functions with the REST Adapter

The REST Adapter can integrate with Oracle Cloud Infrastructure services such as functions and object storage. As an example, assume you need to convert an existing image in object storage to a thumbnail format. You can design an integration in which a REST Adapter connection can take an application/octet-stream for an image, save it to Oracle Cloud Infrastructure object storage, and send the image to an Oracle Cloud Infrastructure function that can create a thumbnail and save it back to object storage.

> **Note:**
>
> You can also directly invoke Oracle Cloud Infrastructure functions and object storage from the integration canvas without using the REST Adapter. See Invoke Oracle Cloud Infrastructure Functions Directly from an Integration with an OCI Function Action and Invoke Oracle Cloud Infrastructure Object Storage from an Integration with an OCI Object Storage Action in *Using Integrations in Oracle Integration 3*.

The following steps provide a high-level overview of creating this type of integration.

1. Go to **Developer Services** > **Functions** in the Oracle Cloud Infrastructure Console, then click an available Oracle Integration instance to view available functions.
2. In the **Invoke endpoint** column, view the function endpoint URL to invoke. For this example, the function is named **node-thumbnail**. The **node-thumbnail** function generates a thumbnail out of an image.
3. Copy the URL. You use this URL to call the function from an integration.
4. Create an integration that accepts an image and generates a thumbnail from that image. When the integration is invoked, both the original image and the generated thumbnail are uploaded to the object storage bucket in Oracle Cloud Infrastructure.
5. Configure the REST Adapter as an invoke connection in the integration.
   a. On the Basic Info page, configure the REST Adapter invoke connection to handle request and response payloads.



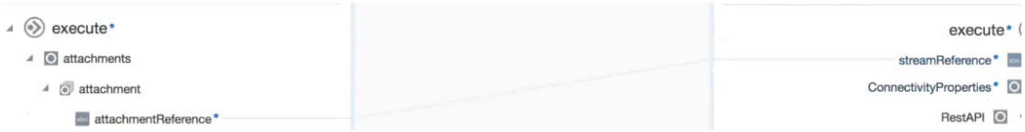
Select any options that you want to configure:

☐ Add and review parameters for this endpoint

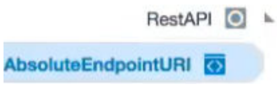
☑ Configure a request payload for this endpoint

☑ Configure this endpoint to receive the response

   b. On the Request page, select **application/octet-stream** as the media type that you want the endpoint to send (this is a binary input stream).
   c. On the Response page, select **application/octet-stream** as the media type to which you want the endpoint to reply (this is also a binary input stream).
6. Configure the mapper as follows:
   a. Map the source **attachmentReference** element to the target **streamReference** for thumbnail generation.



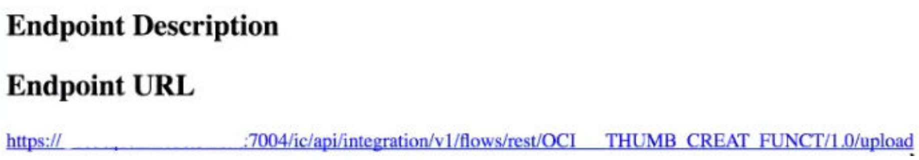
   b. Configure the target **AbsoluteEndpointURI** element to call the **node-thumbnail** function invoke endpoint URL copied in Step 3.



7. Design the remaining portions of the integration.
8. Activate and invoke the integration to call the function.
   a. On the Integrations page, click

      ⚙

      at the far right to show the endpoint URL.
   b. Copy the endpoint URL to an application to invoke the integration, such as `Postman`.

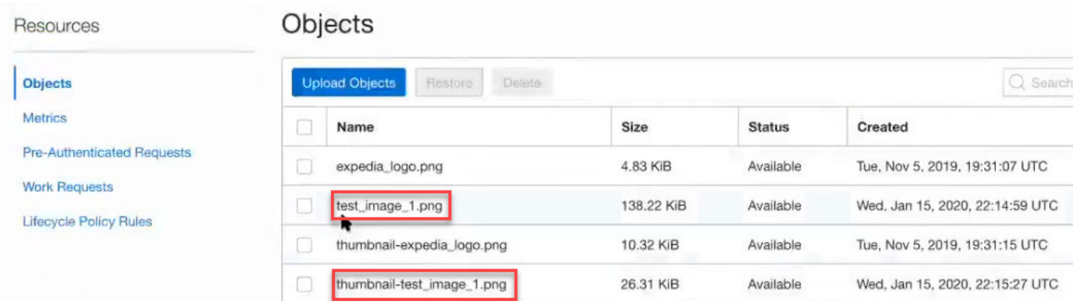


Endpoint Description

Endpoint URL

https:// :7004/ic/api/integration/v1/flows/rest/OCI THUMB CREAT FUNCT/1.0/upload

9. Go to **Object Storage** > **Object Storage** in the Oracle Cloud Infrastructure Console.
10. Click the object storage bucket instance.
11. Refresh the page and note that the original image (**test_image_1.png**) and the generated thumbnail (**thumbnail-test_image_1.png**) are now both displayed.



Objects

	Name	Size	Status	Created
□	expedia_logo.png	4.83 KiB	Available	Tue, Nov 5, 2019, 19:31:07 UTC
□	test_image_1.png	138.22 KiB	Available	Wed, Jan 15, 2020, 22:14:59 UTC
□	thumbnail-expedia_logo.png	10.32 KiB	Available	Tue, Nov 5, 2019, 19:31:15 UTC
□	thumbnail-test_image_1.png	26.31 KiB	Available	Wed, Jan 15, 2020, 22:15:27 UTC

Configure a REST Adapter Trigger Connection to Work Asynchronously

You can configure a trigger REST Adapter connection to work asynchronously in an integration.

For example, assume you have the following use case:

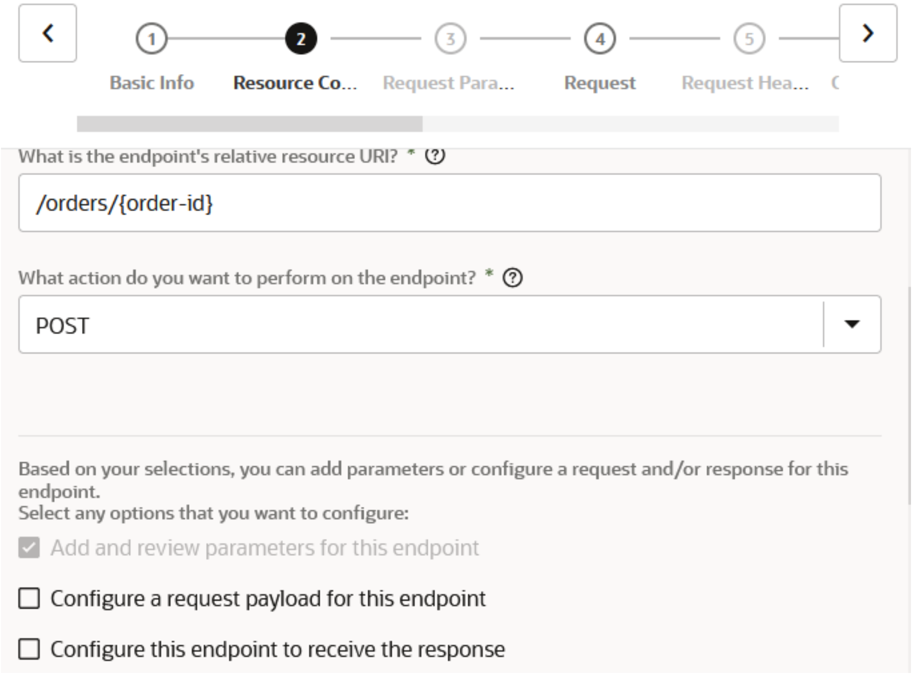
- A parent scheduled integration that runs every three hours.
- A child integration that is executed many times (for example, 300) in a for-each action of the parent integration.

To ensure that this use case runs efficiently, configure the child integration to run asynchronously as follows:

1. Create an application integration for the child integration.
2. Add a REST Adapter as a trigger connection in the integration.

   The Adapter Endpoint Configuration Wizard appears.
3. On the Basic Info page of the wizard, ensure that you configure the page as follows. This ensures that the integration runs asynchronously.

- From the **What action do you want to perform on the endpoint** list, select **POST**.
- From the **Select any options that you want to configure** list, do *NOT* select **Configure this endpoint to receive the response**.



What is the endpoint's relative resource URI? * ⓘ

/orders/{order-id}

What action do you want to perform on the endpoint? * ⓘ

POST

Based on your selections, you can add parameters or configure a request and/or response for this endpoint.
Select any options that you want to configure:

☑ Add and review parameters for this endpoint

□ Configure a request payload for this endpoint

□ Configure this endpoint to receive the response

4. Complete configuration of the REST Adapter.
5. Design the remaining parts of the integration.

   This design ensures that the child integration runs asynchronously.

Create a Keystore File for a Two-Way, SSL-Based Integration

If you need to create an integration that communicates with a two-way, SSL-enabled server, you must create the keystore file required for establishing an Oracle Integration identity to facilitate a two-way, SSL-based integration.

To create an integration that consumes external REST APIs hosted on the two-way, SSL-enabled server, ensure that the server on which the external REST APIs are hosted is enabled for two-way SSL support. Use Java version 1.8 or higher.

You can obtain the client certificate in a variety of ways. Select the method that is best for your environment. For example, you can obtain the certificate directly from many certificate authorities. The steps in this section describe how to generate a certificate signing request (CSR) and have it signed by a well known certificate authority.

Note:

- This section describes how to configure Oracle Integration for use in outbound, two-way SSL integrations. To use Oracle Integration in inbound, two-way SSL integrations, you can use the Oracle Cloud Infrastructure API Gateway. The Oracle Cloud Infrastructure API Gateway is integrated with the Oracle Cloud Infrastructure certificates service. This approach enables you to deliver APIs implemented with Oracle Integration that enforce client mTLS.
- Two-way SSL is not supported for calls to external services through the connectivity agent. Two-way SSL requires direct connectivity from Oracle Integration without the connectivity agent.
- The alias name to provide must match the name provided for the private key entry in the JKS file.

See this blog and Adding mTLS support to API Deployments.

- Commonly Used Terms and Tools
- Commands to Create a Client Certificate with the keytool Utility
- Example: Create a Client Certificate with the keytool Utility
- Manage Certificates with openSSL
- Certificate Management - Two-Way SSL or mTLS

Commonly Used Terms and Tools

Term	Description
Secure socket layer (SSL) and Transport Layer Security (TLS)	SSL and TLS, its successor, are protocols for establishing authenticated and encrypted links between networked computers.
Digital certificate	A data file that holds the cryptographic key provided to an organization or entity by a trusted authority. A simple analogy is a driver’s license. The license uniquely identifies the person to whom it is issued. The license is issued by the DMV, a trusted authority.
Certificate	A public key and private key form a pair used to encrypt and decrypt data. Public keys can be freely given to anyone who needs to securely exchange data. Private keys must never be shared and must be stored securely. If private keys are listed or compromised, the issuing certificate authority must be notified so they can be added to the certificate revocation list.
Certificate authority (or certification authority)	An entity that issues digital certificates. A digital certificate certifies the ownership of a public key by the named subject of the certificate.

Term	Description
Certificate encoding/formats	• Privacy Enhanced Mail (PEM): The full specification of PEM is in RFC 7468. PEM is the most commonly-used X509 certificate format. It's a base64-encoded string enclosed between:`-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` • Distinguished Encoding Rules (DER): Binary Format. Cannot be viewed in an editor. • Public Key Cryptography Standards (PKCS): These are a group of public key cryptography standards devised and published by RSA Security LLC. See https://datatracker.ietf.org/wg/pkix/documents/.
TrustStore	A password-protected repository for trust or public certificates. The default location in Java is `$JAVA_HOME/jre/lib/security/cacerts`. All well known certificate authority root and intermediate certificates are available in the JDK truststore.
Keystore	A password-protected repository to hold client or private certificates. Since this store holds private keys, it is imperative that the store resides in a secure location.
Certificate chain	A certificate chain is an ordered list of certificates ending with the root certificate. For trust to be established, the entire certificate chain is traversed. Each certificate is validated by finding the public key of the next issuing certificate authority or intermediate certificate authority, until the root certificate is reached. Certificate chains are usually cached to validate the certificate locally.

The two most commonly used tools for SSL are the following:

Tool	Description
`keytool`	A JDK utility used to perform CRUD operations on a truststore and keystore and to administer certificates. All the commands require the password that was used to create the store. Consult your Java truststore documentation for the default password.
`openssl`	This is a robust, commercial-grade, full-featured toolkit for the TLS and SSL protocols. It is also a general-purpose cryptography library.

Commands to Create a Client Certificate with the keytool Utility

Commonly used `keytool` commands are as follows.

> **Caution:**
>
> Replace the italicized variables in the commands below with values appropriate to your environment.

Description	Command
List the entire contents of the store	keytool -list -keystore *path_to_the_keystore*
List the contents in the store for a specific alias	keytool -list -keystore *path_to_the_keystore* -alias *alias_name*
View the contents of a certificate	keytool -printcert -v -file *name_of_the_file*
Export a certificate from the store	keytool -export -alias *alias_name* -file *certificate_name* -keystore *path_to_the_store*
Import a new certificate into the store	keytool -import -trustcacerts -file *path_to_the_certificate* -alias *alias_name* -keystore *path_to_the_store*

To create a client certificate:

> **Caution:**
>
> Italicized variables indicate placeholder variables for which you must supply particular values. If you copy the commands below, ensure that you replace the variables shown in italics with values appropriate to your environment.

1. Go to the Java `bin` directory.

   ```
   %JAVA_HOME%/jre/bin
   ```

2. Enter the following command to create a JKS keystore to hold the certificates.

   ```
   keytool -genkey -keyalg RSA -alias alias_name -keystore
   identityKeystore.jks -storepass password_for_the_keystore -validity 360 -
   keysize 2048
   ```

3. When prompted, change the values provided based on your company's security policy.

   ```
   What is your first and last name?
     [Unknown]:  <FQDN>
   What is the name of your organizational unit?
     [Unknown]:  Your_functional_org
   What is the name of your organization?
     [Unknown]:  Company
   What is the name of your City or Locality?
     [Unknown]:  City_name
   What is the name of your State or Province?
     [Unknown]:  State_name
   What is the two-letter country code for this unit?
     [Unknown]:  US
   Is CN=<>, OU=<>, O=<>, L=Redwood Shores, ST=California, C=US correct?
     [no]:  yes
   ```

```
Enter key password for <oicclient>
        (RETURN if same as keystore password):
```

4. Verify the existence of the JKS keystore file.

   ```
   ls
   ```

5. Create a certificate that is ready to be signed.

   ```
   keytool -certreq -alias alias_name -keystore name_of_keystore -storepass
   password -storetype JKS -file name_of_csr_certificate.csr
   ```

6. List the JKS keystore and certificate files in the directory.

   ```
   ls
   ```

7. Validate your CSR file at the following site.

   ```
   https://ssltools.digicert.com/checker/views/csrCheck.jsp
   ```

8. Provide the `.csr` certificate file to a signing authority. A signed certificate and any root and intermediate certificates are signed and returned by the authority. A self-signed certificate can be used for testing, but is not allowed in a production environment.

9. If you have root and intermediate certificates, perform the following substeps. Otherwise, go to Step 10.

   a. If you have a root certificate, enter the following command to import the signed root certificate.

      ```
      keytool -import -keystore keystore_name -file path_to_root_certificate -
      alias root_alias_name
      ```

      The following example is what you see when importing the DigiCert root certificate.

      ```
      Enter keystore password:
      Owner: CN=DigiCert Global Root CA, OU=www.digicert.com, O=DigiCert Inc,
      C=US
      Issuer: CN=DigiCert Global Root CA, OU=www.digicert.com, O=DigiCert
      Inc, C=US
      Serial number: 83be056904246b1a1756ac95991c74a
      Valid from: Thu Nov 09 16:00:00 PST 2006 until: Sun Nov 09 16:00:00 PST
      2031
      Certificate fingerprints:
           MD5:  79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
           SHA1: A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
           SHA256:
      43:48:A0:E9:44:4C:78:CB:26:5E:05:8D:5E:89:44:B4:D8:4F:96:62:BD:26:DB:25:
      7F:89:34:A4:43:C7:01:61
           Signature algorithm name: SHA1withRSA
           Version: 3Extensions:#1: ObjectId: 2.5.29.35 Criticality=false
      AuthorityKeyIdentifier [
      ```

```
KeyIdentifier [
0000: 03 DE 50 35 56 D1 4C BB   66 F0 A3 E2 1B 1B C3
97  ..P5V.L.f.......
0010: B2 3D D1 55                                        .=.U
]
]#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:2147483647
]#3: ObjectId: 2.5.29.15 Criticality=true
KeyUsage [
  DigitalSignature
  Key_CertSign
  Crl_Sign
]#4: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 03 DE 50 35 56 D1 4C BB   66 F0 A3 E2 1B 1B C3
97  ..P5V.L.f.......
0010: B2 3D D1 55                                        .=.U
]
]Trust this certificate? [no]:  yes
Certificate was added to keystore
```

b. If you have an intermediate certificate, enter the following command to import the signed intermediate certificate.

```
keytool -import -keystore keystore_name -file
path_to_intermediate_certificate -alias intermediate_certificate_alias



Enter keystore password: replace_with_strong_password
Certificate was added to keystore
```

10. If you have only a single certificate, enter the following command to import the signed certificate.

```
keytool -import -keystore keystore_name -file path_to_signed_certificate -
alias the_same_alias_used_to_create_the_keystore



Enter keystore password: replace_with_strong_password
Certificate was added to keystore
```

11. Check if all the certificates are in the store.

```
keytool -list -keystore
```

12. Export the public certifcate.

    ```
    keytool -export -alias certificate_alias -keystore identity_keystore -file
    your_public_certificate_filename


    Enter keystore password: replace_with_strong_password
    ```

13. Export the public certificate to provide to the server.

    ```
    keytool -export -alias certificate_alias -keystore identityKeystore.jks -
    file your_public_certificate_filename
    Enter keystore password:
    Certificate stored in file your_public_certificate_filename
    ```

14. Import the new keystore into Oracle Integration as an X509 identity certificate.
15. List the entire contents of the store.

    ```
    keytool -list -keystore path_to_the_keystore
    ```

Example: Create a Client Certificate with the keytool Utility

This section provides an example of how to create a client certificate. It uses actual file names. Replace those names with values appropriate to your environment.

1. Enter the following command to create a JKS keystore to hold the certificates.

    ```
    keytool -genkey -keyalg RSA -alias oicclient -keystore
    identityKeystore.jks -storepass replace_with_strong_password -validity 360
    -keysize 2048
    ```

    Where the following values are entered for this example:

    - `-alias` is the `oicclient` keystore alias.
    - `-keystore` is the `identityKeystore.jks` keystore file.

2. When prompted, change the values provided based on your company's security policy.

    ```
    What is your first and last name?
      [Unknown]:  Joe Smith
    What is the name of your organizational unit?
      [Unknown]:  Development
    What is the name of your organization?
      [Unknown]:  GlobalChips
    What is the name of your City or Locality?
      [Unknown]:  Redwood Shores
    What is the name of your State or Province?
      [Unknown]:  California
    What is the two-letter country code for this unit?
      [Unknown]:  US
    Is CN=<>, OU=<>, O=<>, L=Redwood Shores, ST=California, C=US correct?
      [no]:  yes
    ```

```
Enter key password for oicclient
        (RETURN if same as keystore password):
```

3. Verify the existence of the JKS keystore file.

   ```
   ls
   ```

4. Create a certificate that is ready to be signed.

   ```
   keytool -certreq -alias oicclient -keystore identityKeystore.jks -
   storepass replace_with_strong_password  -storetype JKS -file oicclient.csr
   ```

   Where the following values are entered for this example:

   - `-alias` is the `oicclient` keystore alias.
   - `-keystore` is the `identityKeystore.jks` keystore file.
   - `-file` is the `oicclient.csr` certificate file.

5. List the JKS keystore and certificate files in the directory.

   ```
   ls
   oicclient.csr  identityKeystore.jks
   ```

6. Validate your `.csr` certificate file at the following site.

   ```
   https://ssltools.digicert.com/checker/views/csrCheck.jsp
   ```

7. Provide the `.csr` certificate file to a signing authority. The certificate and any root and intermediate certificates are signed and returned by the authority. A self-signed certificate can be used for testing, but is not allowed in a production environment.

8. If you have root and intermediate certificates, perform the following substeps. Otherwise, go to Step 9.

   a. If you have a root certificate, enter the following command to import the signed root certificate.

      ```
      keytool -import -keystore identityKeystore.jks -file
      DigiCertGlobalRootCA.crt -alias DigiCertCARoot
      ```

      Where the following values are entered for this example:

      - `-keystore` is the `identityKeystore.jks` keystore file.
      - `-file` is the `DigiCertGlobalRootCA.crt` signed root certificate file.
      - `-alias` is the `DigiCertCARoot` alias.

      ```
      Enter keystore password: replace_with_strong_password
      Owner: CN=DigiCert Global Root CA, OU=www.digicert.com, O=DigiCert Inc,
      C=US
      Issuer: CN=DigiCert Global Root CA, OU=www.digicert.com, O=DigiCert
      Inc, C=US
      ```

```
Serial number: 83be056904246b1a1756ac95991c74a
Valid from: Thu Nov 09 16:00:00 PST 2006 until: Sun Nov 09 16:00:00 PST
2031
Certificate fingerprints:
     MD5:  79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
     SHA1: A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
     SHA256:
43:48:A0:E9:44:4C:78:CB:26:5E:05:8D:5E:89:44:B4:D8:4F:96:62:BD:26:DB:25:
7F:89:34:A4:43:C7:01:61
     Signature algorithm name: SHA1withRSA
     Version: 3Extensions:#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 03 DE 50 35 56 D1 4C BB   66 F0 A3 E2 1B 1B C3
97  ..P5V.L.f.......
0010: B2 3D D1 55                                        .=.U
]
]#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:2147483647
]#3: ObjectId: 2.5.29.15 Criticality=true
KeyUsage [
  DigitalSignature
  Key_CertSign
  Crl_Sign
]#4: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 03 DE 50 35 56 D1 4C BB   66 F0 A3 E2 1B 1B C3
97  ..P5V.L.f.......
0010: B2 3D D1 55                                        .=.U
]
]Trust this certificate? [no]:  yes
Certificate was added to keystore
```

b. If you have an intermediate certificate, enter the following command to import the signed intermediate certificate.

```
keytool -import -keystore identityKeystore.jks -file
DigiCertGlobalInterCA.crt -alias DigiCertCAInter
```

Where the following values are entered for this example:

- `-keystore` is the `identityKeystore.jks` keystore file.
- `-file` is the `DigiCertGlobalInterCA.crt` signed intermediate certificate.
- `-alias` is the `DigiCertCAInter` alias.

```
Enter keystore password: replace_with_strong_password
Certificate was added to keystore
```

9. If you have only a single certificate, enter the following command to import the signed certificate.

   ```
   keytool -import -keystore identityKeystore.jks -file
   my_company_signedcert.pem -alias oiclient
   ```

   Where the following values are entered for this example:

   - `-keystore` is the `identityKeystore.jks` keystore file.
   - `-file` is the `my_company_signedcert.pem` signed certificate.
   - `-alias` is the `oiclient` alias.

   ```
   Enter keystore password: replace_with_strong_password
   Certificate was added to keystore
   ```

10. Check if all the certificates are in the store.

    ```
    keytool -list -keystore identityKeystore.jks
    ```

11. Export the public certificate corresponding to the private identity certificate.

    ```
    keytool -export -alias oicclient -keystore identityKeystore.jks -file
    my_company_signedcert.pem
    ```

    Where the following values are entered for this example:

    - `-alias` is the `oicclient` keystore alias.
    - `-keystore` is the `identityKeystore.jks` keystore file.
    - `-file` is the `my_company_signedcert.pem` public certificate file.

    ```
    Enter keystore password: replace_with_strong_password
    Certificate stored in file my_company_signedcert.pem
    ```

12. Import the new keystore (`.jks` file) into Oracle Integration as an X509 identity certificate.
13. List the entire contents of the store.

    ```
    keytool -list -keystore identityKeystore.jks
    ```

Manage Certificates with openSSL

Commonly used `openssl` commands are as follows:

Description	Command
Check a certificate	`openssl x509 -in` *`certificate_name`* `-text -noout`
Get all certificates from a server	`openssl s_client -connect` *`host`*`:`*`ssl_port`* `-showcerts`

Description	Command
Convert a DER format certificate to PEM format	`openssl x509 -inform der -in` *path_to_DER_certificate* `-out` *path_to_PEM_certificate*
Convert a `.pfx` file to a JKS store	`keytool -importkeystore -srckeystore` *path_to_.pfx_file* `-srcstoretype pkcs12 -destkeystore` *path_to_the_jks_file* `-deststoretype JKS -srcstorepass` *pfx_passwd* `-deststorepass` *pfx_passwd*
Convert a `.jks` file to PKCS12 format	`keytool -importkeystore -srckeystore` *path_to_.jks_file* `-destkeystore` *full_path_to_.p12_file*`-srcstoretype JKS - deststoretype PKCS12 -deststorepass` *pkcs12_store_password*
Extract a private key from a `.pfx` file	`openssl pkcs12 -info -in` *path_to_.pfx_file* `-nodes -nocerts -out` *private_key_file_name*
Extract a public certificate from a `.pfx` file	`openssl pkcs12 -in` *path_to_.pfx_file* `-out` *path_to_certificate_file* `-nokeys`

Certificate Management - Two-Way SSL or mTLS

See Debugging SSL/TLS Connections.

To upload an identity certificate:

1. In the navigation pane, select **Home** > **Settings** > **Certificates**.
2. Click **Upload**.
3. Set the alias name to the alias listed in the keystore for the identity certificate. (Use `keytool -list` to see the contents of the keystore.)
4. Make sure the certificate category is set to **Identity**.
5. Upload the client certificate file in JKS format.
6. Enter the keystore and key passwords used to create the JKS store. If there is a mismatch in the passwords, Oracle Integration cannot access the identity certificates.
7. Create a new adapter connection (SOAP Adapter or REST Adapter connection) in Oracle Integration.
8. On the Connections page, select the two-way SSL checkbox and associate the alias required by the connection to use to complete the SSL connection. This alias must match the value that was entered in the Upload Certificate dialog.

To test Mutual TLS authentication (mTLS):

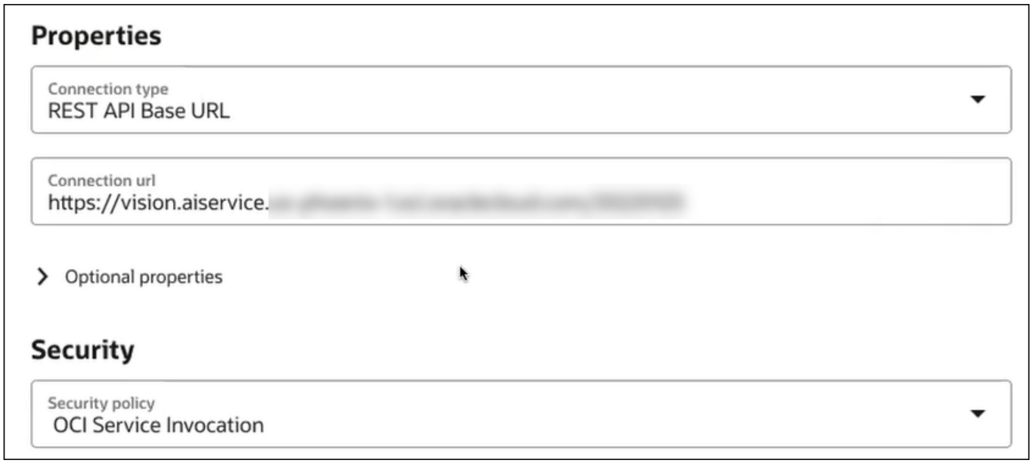
1. Test access to the endpoint from the browser first. Import the client certificate in `.p12` format into the browser of choice.
2. Enter the endpoint in the browser bar and press **Enter**. A message is displayed asking you to use the client certificate that was imported.
3. Follow the prompts in the message. If the certificate is valid, content is loaded in the browser.
4. If the browser test was successful, test the REST/SOAP adapter connection in Oracle Integration.

Access Oracle Cloud Infrastructure Service Resources Using RPST

You can access Oracle Cloud Infrastructure services through use of RPST and the OCI Service Invocation security policy. This example provides a high level overview of designing an integration in which a REST Adapter invoke connection configured with the OCI Service Invocation security policy calls the Oracle Cloud Infrastructure Vision service.

You must complete all prerequisites described in RPST and OCI Service Invocation Security Policy Use.

1. Create an application integration.
2. Create and configure a REST Adapter trigger connection.
3. Create and configure a REST Adapter invoke connection:
   a. Specify the connection type.
   b. Specify the URL of the Oracle Cloud Infrastructure service you want to access. (for this example, the Oracle Cloud Infrastructure Vision service URL is specified). The absolute URL for the Oracle Cloud Infrastructure service is region-specific.
   c. Select the **OCI Service Invocation** security policy. No other user configuration is required.



Properties

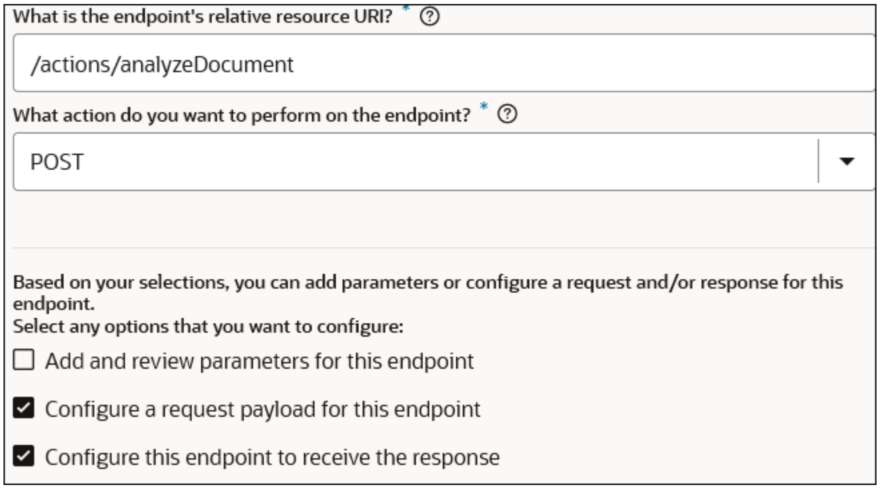
Connection type: REST API Base URL

Connection url: https://vision.aiservice.[redacted]

Security

Security policy: OCI Service Invocation

   d. Test and save the connection.
4. In the integration canvas, add and configure the REST Adapter trigger connection.
5. In the integration canvas, add and configure the REST Adapter invoke connection. For this example, a PDF is uploaded to the Oracle Cloud Infrastructure Vision endpoint to identify the type of document.
   a. On the Basic Info page, specify the service endpoint URL (`/actions/analyzeDocument`), select the action to perform (**POST**), and select the request and response configuration options.



What is the endpoint's relative resource URI? *

/actions/analyzeDocument

What action do you want to perform on the endpoint? *

POST

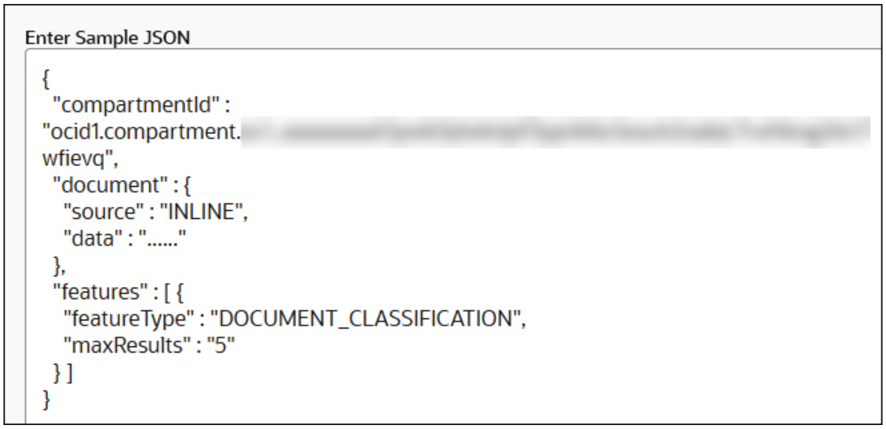
Based on your selections, you can add parameters or configure a request and/or response for this endpoint.
Select any options that you want to configure:

☐ Add and review parameters for this endpoint

☑ Configure a request payload for this endpoint

☑ Configure this endpoint to receive the response

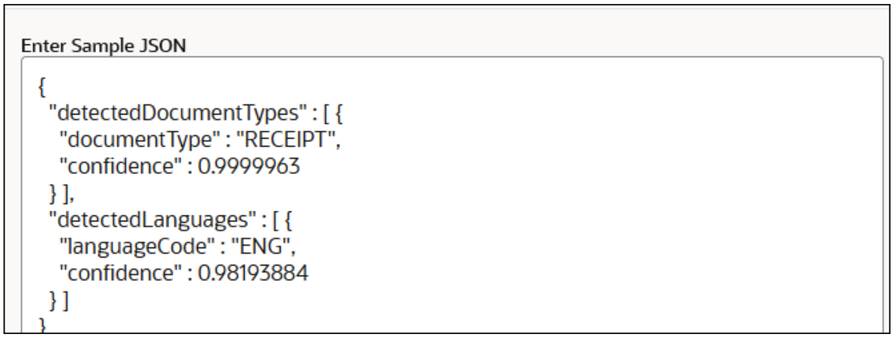
b. On the Request page, specify the media type of the request body (**JSON**) and provide a JSON request sample.



Enter Sample JSON

```
{
  "compartmentId" :
"ocid1.compartment.[redacted]
wfievq",
  "document" : {
    "source" : "INLINE",
    "data" : "......"
  },
  "features" : [ {
    "featureType" : "DOCUMENT_CLASSIFICATION",
    "maxResults" : "5"
  } ]
}
```

c. On the Response page, specify the media type of the response body (**JSON**) and provide a JSON response sample.

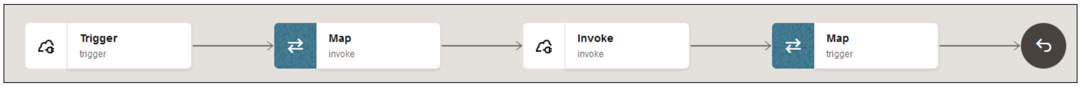


Enter Sample JSON

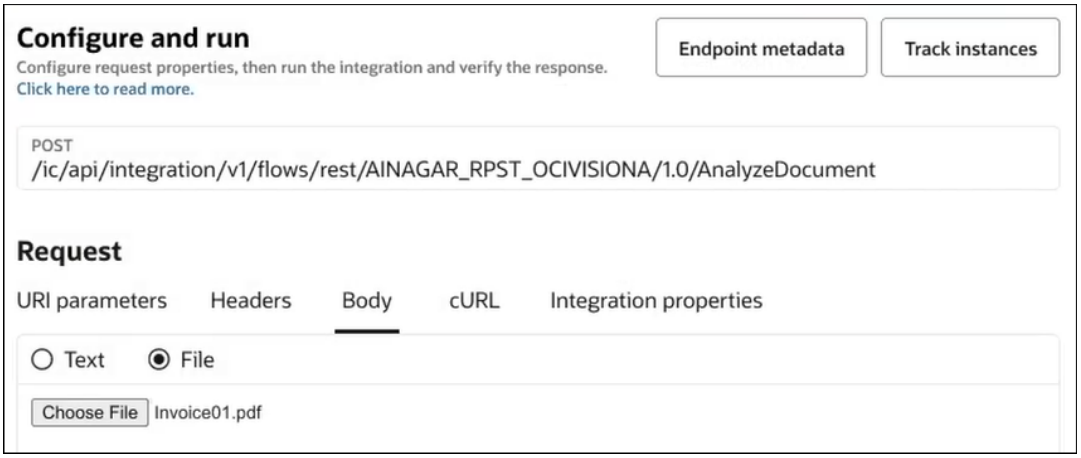
```
{
  "detectedDocumentTypes" : [ {
    "documentType" : "RECEIPT",
    "confidence" : 0.9999963
  } ],
  "detectedLanguages" : [ {
    "languageCode" : "ENG",
    "confidence" : 0.98193884
  } ]
}
```

6. Configure the source-to-target mappings between the two connections.

   When complete, the integration looks as follows.



7. Activate and run the integration.
8. On the Configure and run page, select the PDF to upload to the Oracle Cloud Infrastructure Vision endpoint in the **Body** section and click **Run**.



Configure and run

Configure request properties, then run the integration and verify the response.
Click here to read more.

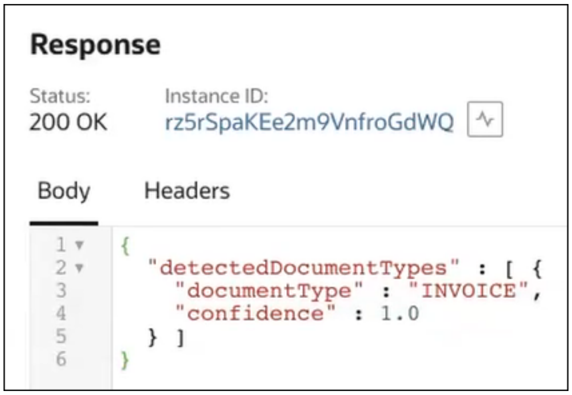
POST
/ic/api/integration/v1/flows/rest/AINAGAR_RPST_OCIVISIONA/1.0/AnalyzeDocument

Request

○ Text ● File

Choose File Invoice01.pdf

The Activity Stream panel opens and shows the movement of messages through the integration. On the Configure and run page, Oracle Cloud Infrastructure Vision detects that the PDF is an invoice, and responds with the following:



Response

Status:
200 OK

Instance ID:
rz5rSpaKEe2m9VnfroGdWQ

```
{
  "detectedDocumentTypes" : [ {
    "documentType" : "INVOICE",
    "confidence" : 1.0
  } ]
}
```

Invoke a Service Provider API with a JWT Assertion

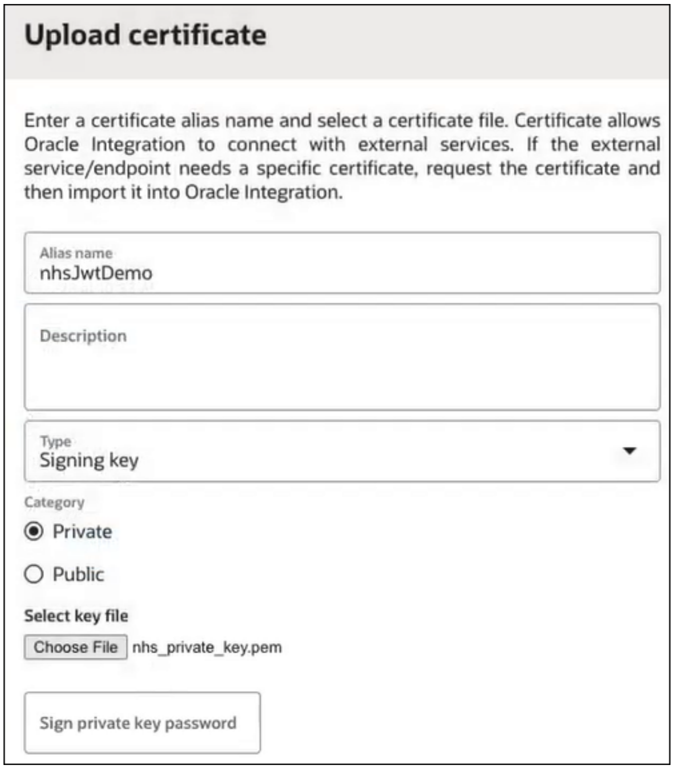
You can invoke the API of a service provider using a JWT user or client assertion security policy in an integration. For this example, a high-level overview is provided of how to call a simple "hello world" API from a service provider using a JWT assertion.

> **Note:**
>
> This use case describes accessing an API of the National Health Service (NHS). The content of the JWT header and JWT payload files uploaded are unique to the standards of the NHS. If accessing an API of a different service provider through a JWT assertion, ensure that your header and payload files conform to the standards of that service provider.

1. Manually create a signing key outside of Oracle Integration.
2. In the navigation pane, click **Settings**, then **Certificates**.
   a. Click **Upload**.
   b. In the **Alias name** field, enter the alias name. You also specify this alias on the Connections page when configuring the JWT assertion security policy. For this example, the following alias is specified.

      ```
      nhsJwtDemo
      ```

   c. From the **Type** list, select **Signing key**.
   d. Click **Private**.
   e. Click **Select key file** to select the signing key you created previously.



Upload certificate

Enter a certificate alias name and select a certificate file. Certificate allows Oracle Integration to connect with external services. If the external service/endpoint needs a specific certificate, request the certificate and then import it into Oracle Integration.

Alias name: nhsJwtDemo

Description:

Type: Signing key

Category

● Private

○ Public

Select key file

Choose File nhs_private_key.pem

Sign private key password

   f. If the private signing key is encrypted, enter the private signing key password.

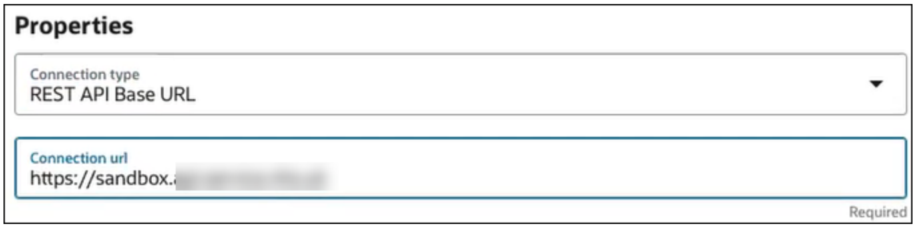
3. In the navigation pane, click **Design**, then **Connections**.

   a. Create a REST Adapter invoke connection, and click **Create**.

      The Connections page is displayed.

   b. In the **Connection type** field, select a connection type.

   c. In the **Connection URL** field, enter the URL of the API to access. For this example, an NHS URL is provided.



Properties

Connection type: REST API Base URL

Connection url: https://sandbox.[redacted]

Required

   d. From the **Security Policy** list, select **OAuth Client Credentials using JWT Client Assertion**. This selection is typically used for application-driven APIs.

   e. In the **Access token URI** field, specify the access token URI for the service provider.

   f. Upload the JWT headers file and JWT payload file.

      You create the content in JSON files. The content of both files is specific to the service provider you want to invoke. For this example, the NHS-formatted header and payload follow this format:

- JWT header file: This file requires an algorithm (`alg`) and a key identifier (`kid`) that is uniquely-generated and associated with the uploaded signing key.

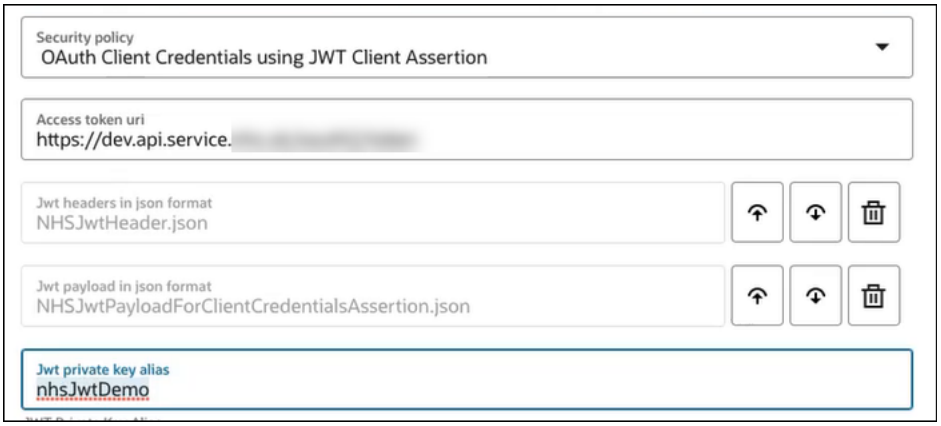
  ```
  : {
    "alg": "RS256",
    "kid": "AfZ4kopskH4V7oe11tRIBDbe4539fie_P",
    }
  ```

- JWT payload file: At a minimum, this file requires the `iss` (issuer of the claim), `sub` (user name subject), `aud` (audience), and `exp` (token expiration time) claims and values (`jti` is optional). These claims are described in your NHS documentation. See Access Tokens and Audit (JWT).

  ```
   {
    "iss": "1445233d3fgd3fbdb30638r536d129fc",
    "sub": "1445233d3fgd3fbdb30638r536d129fc",
    "aud": "https://dev.api.service.nhs.uk/oauth2/token",
    "exp": 1674594057,
    "jti": "b2cdfckd-3gre-2u2d-4150-2062f10cfbd4"
  }
  ```

Additional claims, including custom claims, can also be included in both files. See the documentation provided by the service provider for instructions on how to configure these files.

g. For the JWT private key alias, enter the same name you specified when uploading the signing key certificate in Step 1. This name is used to generate the JWT assertion.



Security policy
OAuth Client Credentials using JWT Client Assertion

Access token uri
https://dev.api.service.[redacted]

Jwt headers in json format
NHSJwtHeader.json

Jwt payload in json format
NHSJwtPayloadForClientCredentialsAssertion.json

Jwt private key alias
nhsJwtDemo

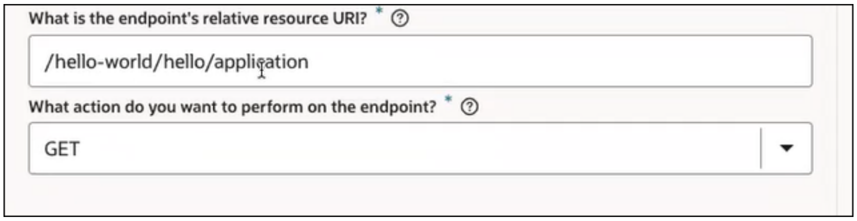
h. Click **Optional security** if you want to specify optional scopes or access token request values. For this example, none are specified. The way to specify the scope and access token request values varies from service providers. Some service providers require values. See Variations of JWT Usage by Service Providers.

i. Test the connection.

   You are ready to create an integration that uses this connection to call the NHS API.

4. In the navigation pane, click **Design**, then **Integrations**.
5. Create an application integration to call the NHS API and receive an "echo" in response.
6. Design the integration. For example, add and configure a REST Adapter trigger connection.

7. Add a REST Adapter as an invoke connection to call the NHS API.
   a. On the Basic Info page of the Adapter Endpoint Configuration Wizard, specify the relative resource URI. For this example, a "hello world" API provided by the NHS is specified. This API calls back to the integration after the JWT token is successfully validated by the NHS service provider.
   b. Select the action to perform.



What is the endpoint's relative resource URI? *

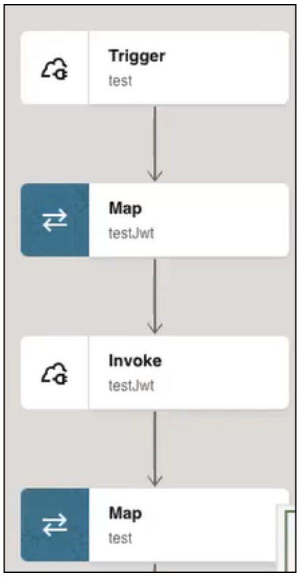
/hello-world/hello/application

What action do you want to perform on the endpoint? *

GET

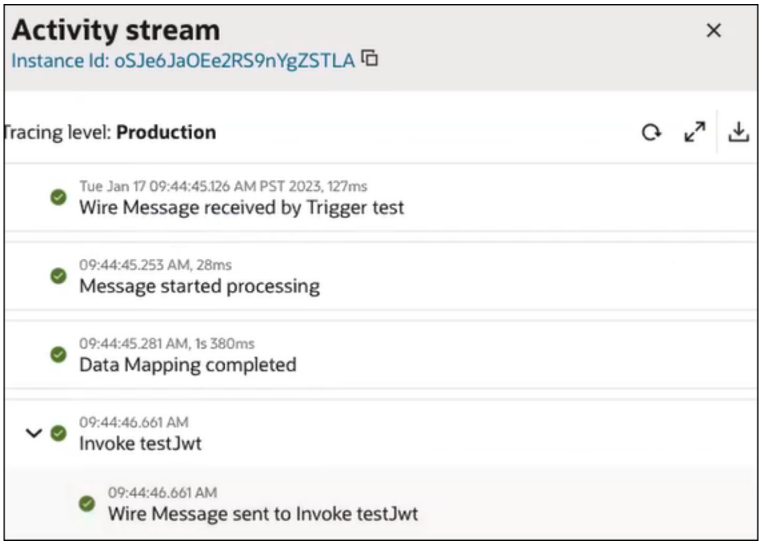
   c. Complete the remaining fields of the wizard.
8. Complete the remaining design of the integration.

   When complete, the integration looks as follows:



9. Activate the integration.
10. Hover over the row of the integration on the Integrations page.
11. Select **Actions** ..., then select **Run**.
12. Run the integration.

    The activity stream provides details, including the response sent back.



Activity stream
Instance Id: oSJe6JaOEe2RS9nYgZSTLA

Tracing level: **Production**

Tue Jan 17 09:44:45.126 AM PST 2023, 127ms
Wire Message received by Trigger test

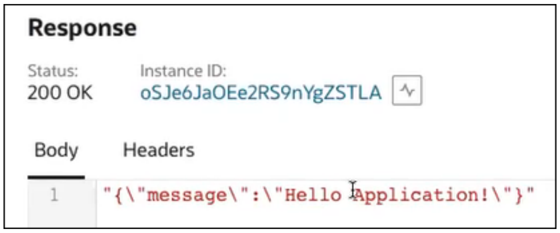
09:44:45.253 AM, 28ms
Message started processing

09:44:45.281 AM, 1s 380ms
Data Mapping completed

09:44:46.661 AM
Invoke testJwt

09:44:46.661 AM
Wire Message sent to Invoke testJwt

In the Configure and run page, the response message indicates that the NHS service provider validated the token and responded back with a `Hello Application!` message.



Response

Status: 200 OK
Instance ID: oSJe6JaOEe2RS9nYgZSTLA

```
"{\"message\":\"Hello Application!\"}"
```

See JWT Assertion Support for Outbound Invocations.

Propagate User Identity from the REST Adapter to Oracle NetSuite with Custom JWT Claims

You can propagate the user identity between the REST Adapter and Oracle NetSuite with custom JWT claims. This section provides an overview of designing an integration and propagating the user identity at runtime in Postman.

You must complete prerequisites before you can use this feature. See Prerequisites for Identity Propagation Between the REST Adapter and Oracle NetSuite.

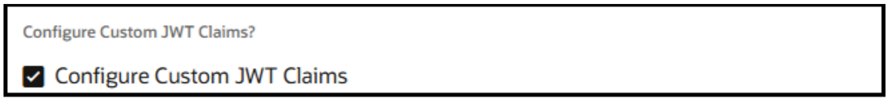
- Create Connections
- Create and Design an Application Integration
- Create the Runtime Token ID and Token Secret Values in Oracle NetSuite
- Add the Custom Claim APIs to the Oracle Cloud Infrastructure Identity and Access Management Instance of Oracle Integration
- Configure the Client Token to Use
- Run the Integration Using the Runtime Token ID and Token Secret Values

Create Connections

1. Create a REST Adapter trigger connection.
2. Create an Oracle NetSuite Adapter invoke connection with the Token-Based Authentication security policy. See Configure Connection Security in *Using the Oracle NetSuite Adapter with Oracle Integration 3*. During configuration, you specify the token ID and token secret values.

Create and Design an Application Integration

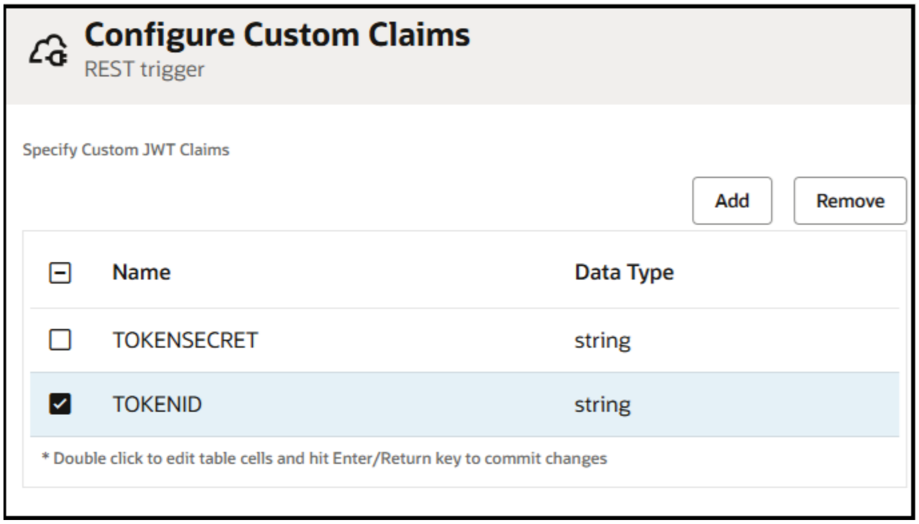
1. Create an application integration.
2. Add and configure the REST Adapter trigger connection in the integration canvas. The identity propagation portion of the configuration process is described below.
   a. On the Resource Configuration page, select **Configure Custom JWT Claims**.



Configure Custom JWT Claims?

☑ Configure Custom JWT Claims

   b. On the Custom Claims page, add custom JWT claims for token secret and token ID. These claims must also be defined in the Oracle Cloud Infrastructure Identity and Access Management instance.



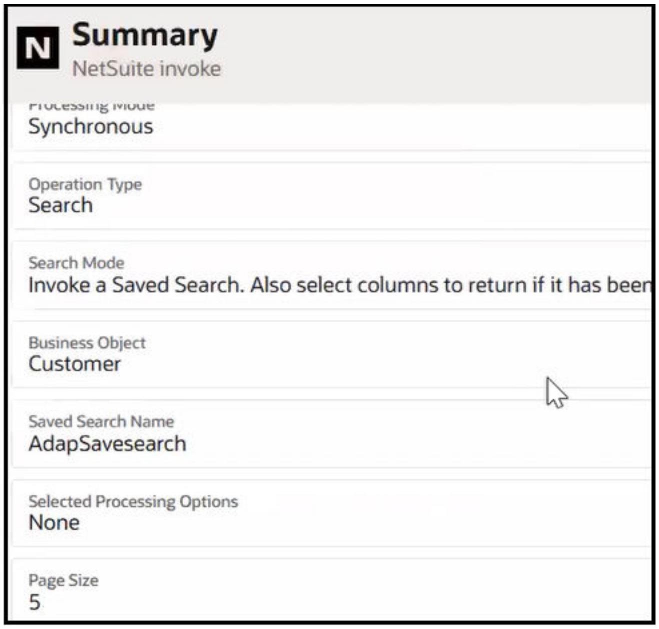
Configure Custom Claims
REST trigger

Specify Custom JWT Claims

⊟	Name	Data Type
□	TOKENSECRET	string
☑	TOKENID	string

* Double click to edit table cells and hit Enter/Return key to commit changes

3. Add and configure the Oracle NetSuite Adapter invoke connection in the integration canvas. Configure the adapter to address your business needs. There are *no* identity propagation settings that require configuration. For this example, the Summary page for the Oracle NetSuite Adapter looks as follows upon completion.



Summary
NetSuite invoke

Processing Mode
Synchronous

Operation Type
Search

Search Mode
Invoke a Saved Search. Also select columns to return if it has been

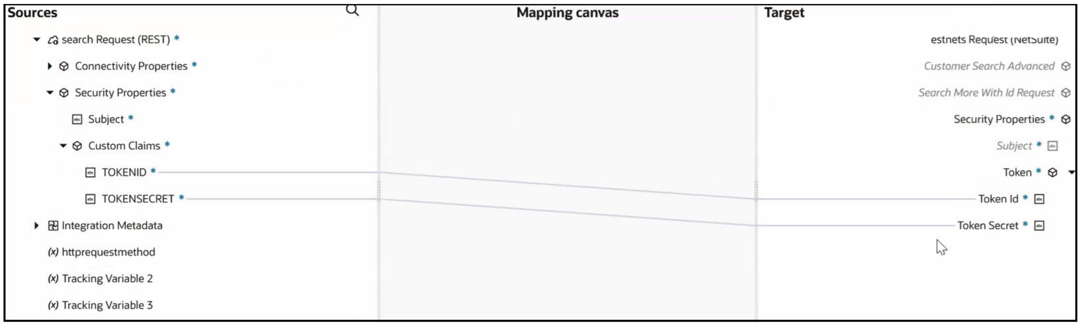
Business Object
Customer

Saved Search Name
AdapSavesearch

Selected Processing Options
None

Page Size
5

4. Open the mapper and expand the **Sources** and **Target** sections.
5. In the source **Custom Claims** section under **Security Properties**, map the previously defined **TOKENID** and **TOKENSECRET** custom JWT claim elements to the target **Token Id** and **Token Secret** elements.
   The target **Security Properties** section automatically appears whenever you create a new endpoint connection to Oracle NetSuite. The **Custom Claims** mappings enable the user identity to be propagated and automatically authenticated only in Oracle NetSuite. The target **Token Id** and **Token Secret** elements enable you to override the client ID and client secret values configured for the Token-Based Authentication security policy of the Oracle NetSuite Adapter at design-time.



Note:

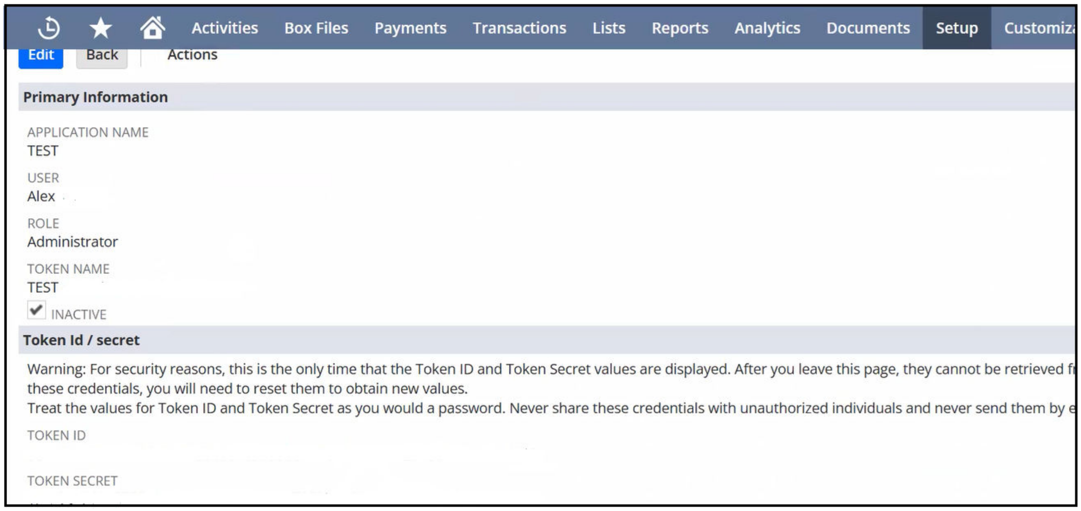
- You must use the **Custom Claims** elements for user propagation with Oracle NetSuite. Do *not* use the **Subject** elements.
- To make the **Security Properties** section visible in an already-created Oracle NetSuite Adapter invoke connection, open the Adapter Endpoint Configuration Wizard, click **Continue** on each page, then click **Finish**.

6. Complete integration design.
7. Create a business identifier, and activate the integration.

Create the Runtime Token ID and Token Secret Values in Oracle NetSuite

You must create the token ID and token secret values that you want to specify at runtime when propagating the user identity to Oracle NetSuite. These values are specified for the token ID and token ID parameters at runtime.

1. Sign in to Oracle NetSuite.
2. Click **Home** in the main menu.
3. Scroll down on the left side of the page.
4. Click **Manage Access Tokens** under the **Settings** section.
5. Click **New My Access Token**.
6. Select **ICS Integration Record** from the **Application Name** list, and click **Save**.
   The token secret and token ID values to specify at runtime are created (obfuscated in the following image for security reasons).



Primary Information

APPLICATION NAME
TEST

USER
Alex

ROLE
Administrator

TOKEN NAME
TEST

☑ INACTIVE

Token Id / secret

Warning: For security reasons, this is the only time that the Token ID and Token Secret values are displayed. After you leave this page, they cannot be retrieved f
these credentials, you will need to reset them to obtain new values.
Treat the values for Token ID and Token Secret as you would a password. Never share these credentials with unauthorized individuals and never send them by e

TOKEN ID

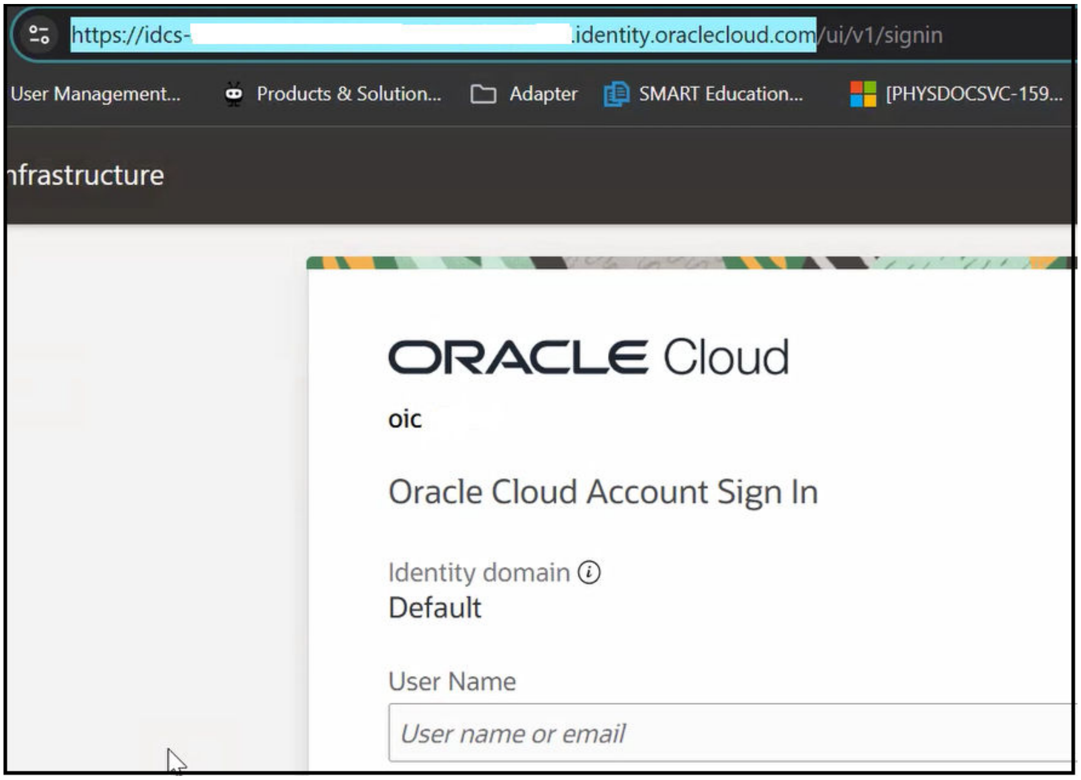
TOKEN SECRET

7. Copy the token ID and token secret values for specifying at runtime.

Add the Custom Claim APIs to the Oracle Cloud Infrastructure Identity and Access Management Instance of Oracle Integration

You must add the custom claim APIs to the Oracle Cloud Infrastructure Identity and Access Management instance of Oracle Integration. See Prerequisites for Identity Propagation Between the REST Adapter and Oracle NetSuite. The following steps describe how to make this association in Postman at runtime.

1. Open a tool to run the integration. For this example, Postman is used.
2. Go to the Oracle Cloud Infrastructure Identity and Access Management login page and copy the URL that begins with `idcs-`.



https://idcs-[redacted].identity.oraclecloud.com/ui/v1/signin

ORACLE Cloud

oic

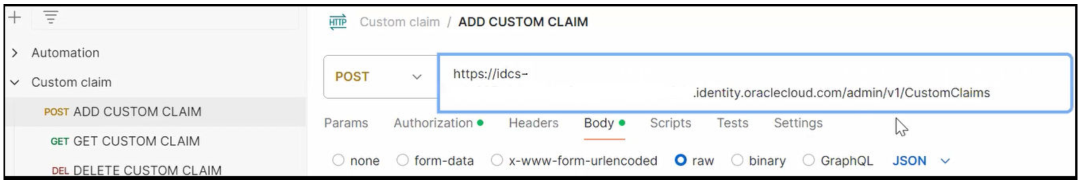
Oracle Cloud Account Sign In

Identity domain ⓘ
Default

User Name
User name or email

3. Paste the URL into the **Post** field in Postman.
4. Add `/admin/v1/CustomClaims` to the end. This action associates the custom claim APIs with the Oracle Cloud Infrastructure Identity and Access Management instance of Oracle Integration.

   ```
   https://idcs-URL_value.identity.oraclecloud.com/admin/v1/CustomClaims
   ```



POST https://idcs-[redacted].identity.oraclecloud.com/admin/v1/CustomClaims

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

5. Under **Body**, click **JSON**, paste in the entire JSON body, and make the following updates.
   - `schemas`: Enter the custom claims schema.
   - `name`: Enter the exact name of the token ID custom claim you added on the Custom Claims page when configuring the REST Adapter in Create and Design an Application Integration. The names must match. If you change the name on the Custom Claims page of the REST Adapter, you must change it here to match.
   - `value`: Enter the token ID value you obtained from Oracle NetSuite in Create the Runtime Token ID and Token Secret Values in Oracle NetSuite.
   - `expression`: Enter `false`.
   - `mode`: Enter `request`.

- `tokenType`: Enter `AT`.
- `allScopes`: Enter `false` because the token ID is only applicable to a specific scope.
- `scopes`: Enter the scope value in which the custom JWT claim is available. You selected the scope value when creating the confidential application in the Oracle Cloud Infrastructure Console. The scope is used when you click **Test Custom Claims** to run the integration. See Prerequisites for Identity Propagation Between the REST Adapter and Oracle NetSuite.

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:idcs:CustomClaim"
    ],
    "name": "TOKENID",
    "value": "token_ID_value",
    "expression": false,
    "mode":  "request",
    "tokenType":  "AT".
    "allScopes":  false,
    "scopes": [
    "https://1FAE03234F573435FE325EC543F49771.integration.us-
region-1.ocp.oraclecloud.com
        :443urn:opc:resource:consumer::all"]
}
```

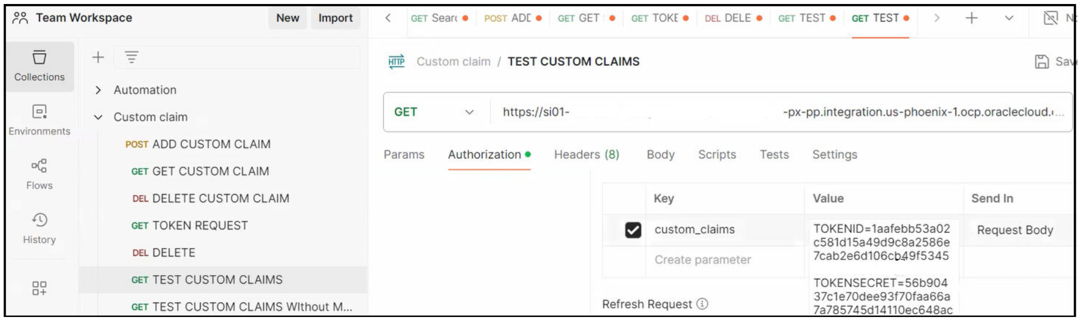
Configure the Client Token to Use

You must specify additional parameter values to run your integration.

1. Click **Test Custom Claims**.
2. Click the **Authorization** tab, then **Body**, and scroll down to **Configure New Token** to specify the following values. The client ID, client secret, and scope values are used when triggering the integration flow using the client credentials confidential application for the trigger connection.
   - **Token name**: Enter the token name. The token name can be anything. The token name is provided so that it can be re-used for other requests.
   - **Grant type**: Select **Client Credentials** from the list. This is the type you selected when creating your confidential application. Custom JWT claims work with this grant type.
   - **Access Token URL**: Specify the access token URL. This URL is used to create the access token. This is part of the OAuth client credentials security policy.
   - **Client ID**: Specify the client ID of the user with administrator privileges to the Oracle Cloud Infrastructure Identity and Access Management instance. This is *not* the client ID you obtained in Create the Runtime Token ID and Token Secret Values in Oracle NetSuite.
   - **Client Secret**: Specify the client secret of the user with administrator privileges to the Oracle Cloud Infrastructure Identity and Access Management instance. This is *not* the client secret you obtained in Create the Runtime Token ID and Token Secret Values in Oracle NetSuite.
   - **Scope**: `urn:opc:idm:_myscopes_`. This is the value added from the custom claims specified in Oracle Cloud Infrastructure Identity and Access Management by the user with administration privileges.
   - **Client Authentication**: Select **Send as Basic Auth Header** from the list.

Run the Integration Using the Runtime Token ID and Token Secret Values

1. In the **Get** field under **Authorization**, enter the URL of the integration to execute.
2. Click the **custom_claims** key, and enter the runtime token ID and token secret values you obtained in Create the Runtime Token ID and Token Secret Values in Oracle NetSuite. The actual token ID and token secret names (for this example, **TOKENIN** and **TOKENSECRET**) must match with those names you specified on the Custom Claims page during design time. The **custom_claims** key is optional and can be added when making an access token request.

   Custom claim / TEST CUSTOM CLAIMS

   GET https://si01- -px-pp.integration.us-phoenix-1.ocp.oraclecloud.…

	Key	Value	Send In

☑	custom_claims	TOKENID=1aafebb53a02c581d15a49d9c8a2586e7cab2e6d106cb49f5345
		TOKENSECRET=56b90437c1e70dee93f70faa66a7a785745d14110ec648ac
			Request Body

	Create parameter		

3. Click **Get New Access Token**, then **Proceed**, and then **Use Token**.
   If the connection to the Oracle NetSuite endpoint is successful, the user is available. The token secret and token ID of the user were successfully propagated to Oracle NetSuite.

   If you specify an incorrect token ID or token secret, that user is not available and you receive the following error. Verify that the values you entered for client ID and client secret are correct.

   ```
   Invalid login attempt
   ```

6 Troubleshoot the REST Adapter

Review the following topics to learn about troubleshooting issues with the REST Adapter.

Topics:

ORABPEL-15235 Translation Failure Error Occurrence

If the string length for fields in a JSON payload exceeds 20 MB, a REST Adapter trigger connection fails with the following error:

```
ORABPEL-15235 Translation Failure. Failed to translate JSON to XML. String
length (20038094)
exceeds the maximum length (20000000) The incoming data does not conform to
the NXSD schema.
Please correct the problem
```

As a workaround, remodel your integration to work around this field size restriction. Some recommendations are:

- Write a file to an external SFTP server for processing outside of Oracle Integration.
- Add an attachment as a stream when invoking the endpoint rather than sending the content as a Base64-encoded string.

Failed REST Adapter Invoke Connection Retries Three Times Every 30 Seconds with a 504 Timeout Error

A REST Adapter connection attempting to invoke an external REST service retries three times approximately every 30 seconds upon receiving a `504` gateway timeout error. This behavior is by design and cannot be changed.

The REST Adapter is designed to support the following types of retries:

- Three retries for a `504` error (from an external service) for all cases.
- Three retries for a `502` error (from an external service), but only for GET operations.
- Three retries for SSL and adapter-marked retryable exceptions (with 500 milliseconds of sleep between each call).

Troubleshoot RPST and OCI Service Invocation Security Policy Issues

Learn about troubleshooting configuration issues when using the Resource Principal Session Token (RPST) and the OCI Service Invocation security policy.

Error	Cause of Error and Resolution

A `404 Not Found` error occurs with the following message:
`[CDATA[{\"code\":\"NamespaceNotFound\",\"message\":\"You do not have authorization to perform this request,`
`or the requested resource could not be found.\"}.The HTTP 404, 404 Not Found, and 404 error message is a Hypertext`
`Transfer Protocol (HTTP) standard response code, in computer network communications, to indicate that the client`
`was able to communicate with a given server, but the server could not find the resource that was requested. Carefully`
`re-examine the target endpoint that is being called. ]]>`
	You used the OCI Service Invocation security policy without setting the correct policy. For this example, the Get Object Storage Namespace operation worked, but other operations failed because the policy was not set to access or modify a compartment.
	Set your policy to perform the required operations. See RPST and OCI Service Invocation Security Policy Use.

A `409 Not Found` error occurs with the following message:
`[CDATA[{"code":"BucketAlreadyExists","message":"Either the bucket 'test-rpst-OICServiceInstance-phx-NoPermission'`
`in namespace 'axkbv4jfb37h' already exists or you are not authorized to create it"}.This error usually indicates that the`
`request submitted by Oracle Integration Cloud can not be completed because it conflicts with some rule already established`
`by the service. This is usually a functional error. If the call to the target service using a cURL request is successful`
`then contact oracle support with the details.`
	A policy was created that only lets you read buckets in a compartment:
	`allow dynamic-group rpst_oic-prod-phx-123_Dynamic_Group to read buckets in compartment oic-dev:pp-phx-123-comp`
	If you try to create a bucket in the compartment, it fails because the policy is not set to perform the required operation.
	Create a policy that allows you to write to the bucket. See RPST and OCI Service Invocation Security Policy Use.

Error	Cause of Error and Resolution

A `404 Not Found` error occurs with the following message:
 	
`<!`
`[CDATA[{\"code\":\"NamespaceNotFound\",\"message\`
`":\"You do not have authorization to perform`
`this request, or`
`the requested resource could not be`
`found.\"}.The HTTP 404, 404 Not Found, and 404`
`error message is a Hypertext Transfer`
`Protocol (HTTP) standard response code, in`
`computer network communications, to indicate`
`that the client was able to communicate`
`with a given server, but the server could not`
`find the resource that was requested. Carefully`
`re-examine the target endpoint`
`that is being called. ]]>`
	The Get Object Storage Namespace operation works, but other operations fail due to the dynamic group being specified as follows:
	
	`instance.id =`
	`service_instance_ocid`
	
	RPST expects the resource ID, not the instance ID, to be specified. Specify the resource ID and assign it the client ID of the OAuth application of your Oracle Integration instance when creating the dynamic group:
	
	`resource.id = '`*`client_ID`*`'`
	
	See RPST and OCI Service Invocation Security Policy Use.

A `404 Not Found` error occurs with the following message:
 	
`<!`
`[CDATA[{\"code\":\"NamespaceNotFound\",\"message\`
`":\"You do not`
`have authorization to perform this request, or`
`the requested resource could`
`not be found.\"}.The HTTP 404, 404 Not Found,`
`and 404 error message is a`
`Hypertext Transfer Protocol (HTTP) standard`
`response code, in computer`
`network communications, to indicate that the`
`client was able to communicate`
`with a given server, but the server could not`
`find the resource that was`
`requested. Carefully re-examine the target`
`endpoint that is being called. ]]>`
	The Get Object Storage Namespace operation works, but other operations fail due to the dynamic group being specified as follows:
	
	`resource.id =`
	`service_instance_OCID`
	
	Assign the client ID of the OAuth application of your Oracle Integration instance to the resource ID when creating the dynamic group:
	
	`resource.id = '`*`client_ID`*`'`
	
	See RPST and OCI Service Invocation Security Policy Use.

Multipart Form-Data Endpoint Invocation Fails When Media Type is null

Ensure that the attachment properties - attachment content type is mapped with an appropriate value. The sample documentation for request mapping provides information.

See Map to Construct the Payload for an External REST API that Accepts multipart/form-data as the Content Type.

Convert a Private Key from PKCS8 to RSA (PKCS1) Format for the OCI Signature Version 1 Security Policy

Private keys downloaded from the Oracle Cloud Infrastructure Console are in PKCS8 format. The OCI Signature Version 1 security policy available with the REST Adapter only supports reading of the private key in RSA format (PKCS1 format).

If you receive the following error, you must convert the private key from PKCS8 to RSA (PKCS1) format.

```
CASDK-0005: A connector specific exception was raised by the application.
java.lang.ClassCastException: org.bouncycastle.asn1.pkcs.PrivateKeyInfo
cannot be cast to org.bouncycastle.openssl.PEMKeyPair;
org.bouncycastle.asn1.pkcs.PrivateKeyInfo cannot be cast to
org.bouncycastle.openssl.PEMKeyPair
```

1. Convert the private key with the following command:

   ```
   openssl pkey -in private_key_in_pkcs8_format.pem -traditional >
   new_converted_file.key
   ```

   For example:

   ```
   openssl pkey -in pkcs8.pem -traditional > rsa.key
   ```

   An example of a PKCS8-formatted private key file:

   ```
   -----BEGIN PRIVATE KEY-----
   contents
   -----END PRIVATE KEY-----
   ```

   An example of an RSA (PKCS1)-formatted file after the conversion:

   ```
   -----BEGIN RSA PRIVATE KEY-----
   contents
   -----END RSA PRIVATE KEY-----
   ```

HTTP Error Response for Pre-20.4.2 Connections is Not Compliant with the OpenAPI Specification

The HTTP error response returned by integrations created prior to release 20.4.2 (November 2020 quarterly release) that include a REST Adapter-based trigger connection does not strictly conform to the OpenAPI specification.

The error response returned has an `o:` prefixed to certain keys. For example:

```
{
  "type" :
"http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1",
  "title" : "Internal Server Error",
  "detail" : "An internal error occurred while processing the request. Please
see the fault details for the nested error details.",
  "o:errorCode" : "400",
  "o:errorDetails" : [ {
    "type" :
"http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.5",
    "instance" : "<![CDATA[{}.The HTTP 404, 404 Not Found, and 404 error
message
is a Hypertext Transfer Protocol (HTTP) standard response code, in computer
network communications, to indicate that the client was able to communicate
with a given server, but the server could not find the resource that was
requested.
Carefully re-examine the target endpoint that is being called. ]]>",
    "title" : "Not Found",
    "o:errorPath" : "<![CDATA[GET http://api.zippopotam.us/us/9021011
returned a response status of 404 Not Found]]>",
    "o:errorCode" : "404"
  } ]
}
```

With the November 2020 release (20.4.2), the HTTP error response format has been modified for newly created integrations to be compliant with the OpenAPI specification. The `o:` previously prefixed to certain keys has been removed, as shown in the following message.

However, note that all existing, pre-20.4.2 integrations continue to have the same error response shown above, even when those integrations are modified.

```
{
  "type" :
"http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1",
  "title" : "Internal Server Error",
  "detail" : "An internal error occurred while processing the request. Please
see the fault details for the nested error details.",
  "errorCode" : "404",
  "errorDetails" : [ {
    "type" :
"http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.5",
    "instance" : "<![CDATA[{}.The HTTP 404, 404 Not Found, and 404 error
message
is a Hypertext Transfer Protocol (HTTP) standard response code, in computer
network communications, to indicate that the client was able to communicate
with a given server, but the server could not find the resource that was
requested.
Carefully re-examine the target endpoint that is being called. ]]>",
    "title" : "Not Found",
    "errorPath" : "<![CDATA[GET http://api.zippopotam.us/us/9021011 returned a
response status of 404 Not Found]]>",
    "errorCode" : "404"
  } ]
}
```

REST Services that Return Multiple Successful Responses

The REST Adapter can be configured for only a single type of response. A service that returns multiple responses, even with different HTTP success status codes, is not supported. All except for the configured response type result in an `APIInvocationError`. You can catch the resulting error using a scope action and a fault handler if the fault is not required in the integration.

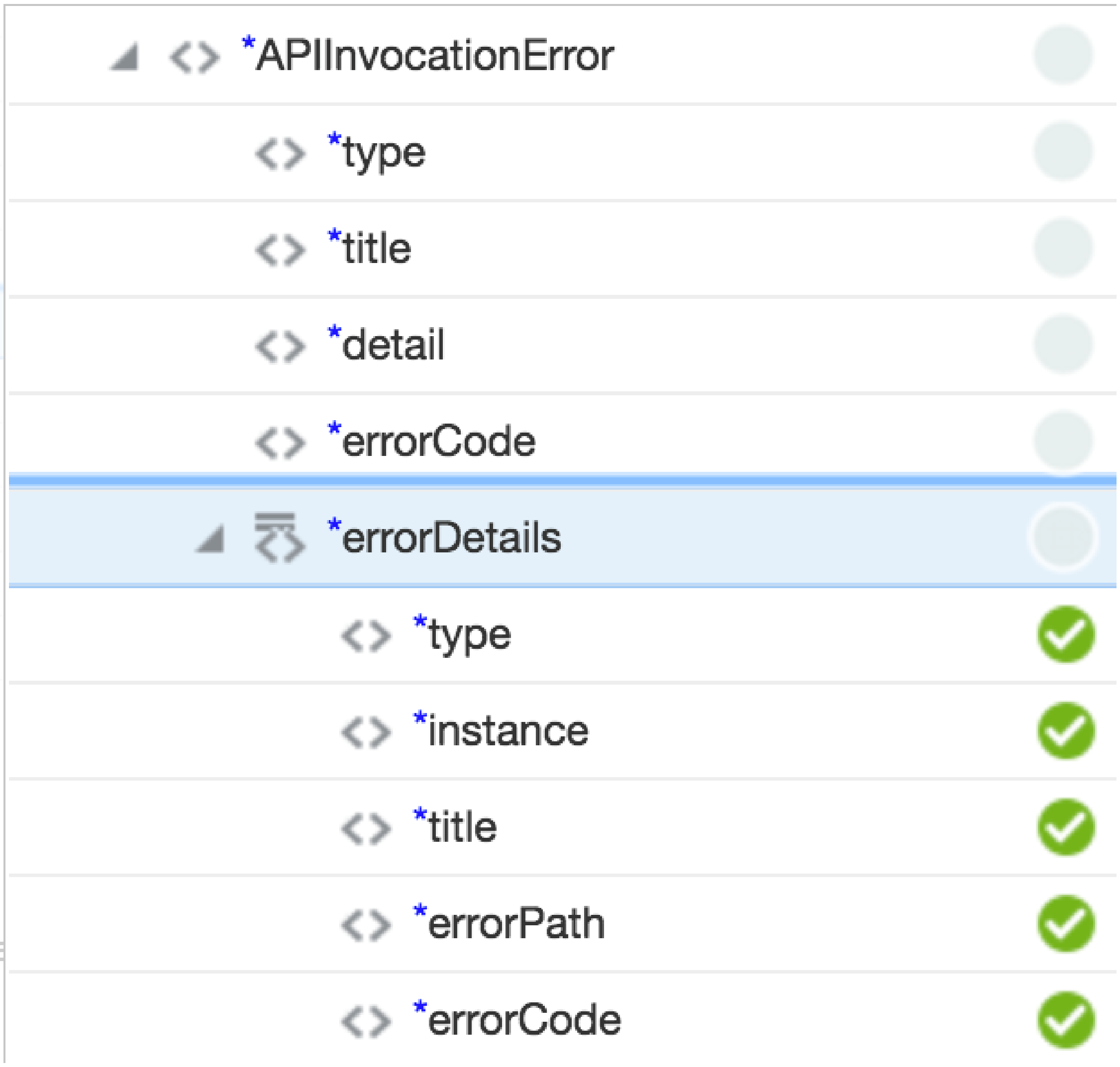
Error Handling with the REST Adapter

The REST Adapter uses the following strategy to handle errors in the invoke (outbound) and trigger (inbound) directions.

Error Handling in the Invoke (Outbound) Direction

The REST Adapter in the invoke (outbound) direction returns a standard `APIInvocationError` for any HTTP response that it receives with an error code. In addition, it also produces an `APIInvocationError` if a processing error occurs within the REST Adapter while preparing the request, calling the endpoint, or handling the response.

The format of the `APIInvocationError` in the mapper is as follows.



The `errorDetails` section contains the actual cause.

You can handle the `APIInvocationError` with a fault handler in the integration.

Error Handling in the Trigger (Inbound) Direction

The REST Adapter in the trigger (inbound) direction exposes an HTTP endpoint that HTTP clients can request for using an HTTP request, and returns an HTTP response.

If successful, the REST Adapter returns a success response. The REST Adapter returns an error response with an HTTP status belonging to the error family of codes depending on the situation. The following table describes the possible cause and the REST Adapter response.

Condition	HTTP Status	Details
Invalid client request	4*xx*	There are several conditions that can cause client side failures, including: • An invalid resource URL • Incorrect query parameters • An unsupported method type • An unsupported media type • Bad data
Downstream processing errors	5*xx*	All other errors that can occur within the integration, including: • An invalid target • An HTTP error response • General processing errors

In addition, the REST Adapter also returns an error response with additional details about the error and possible steps for troubleshooting. The standard error response format is returned according to the configured response media type. The following is a sample JSON response structure:

```
{
  "type" : "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1",
  "title" : "Internal Server Error",
  "detail" : "An internal error occurred while processing the request. Please
see the fault details for the nested error details.",
  "o:errorCode" : "500",
  "o:errorDetails" : [ {
    "type" : "http://www.w3.org/Protocols/rfc2616/rfc2616-
sec10.html#sec10.4.1",
    "instance" : "{\n   \"error_message\" : \"Invalid request. Missing the
'origin' parameter.\",\n   \"routes\" : [],\n
    \"status\" : \"INVALID_REQUEST\"\n}\n",
    "title" : "Bad Request",
    "o:errorPath" : "GET http://maps.googleapis.com/maps/api/directions/json?
destination=Montreal returned a response status of
      400 Bad Request",
    "o:errorCode" : "APIInvocationError"
  } ]
}
```

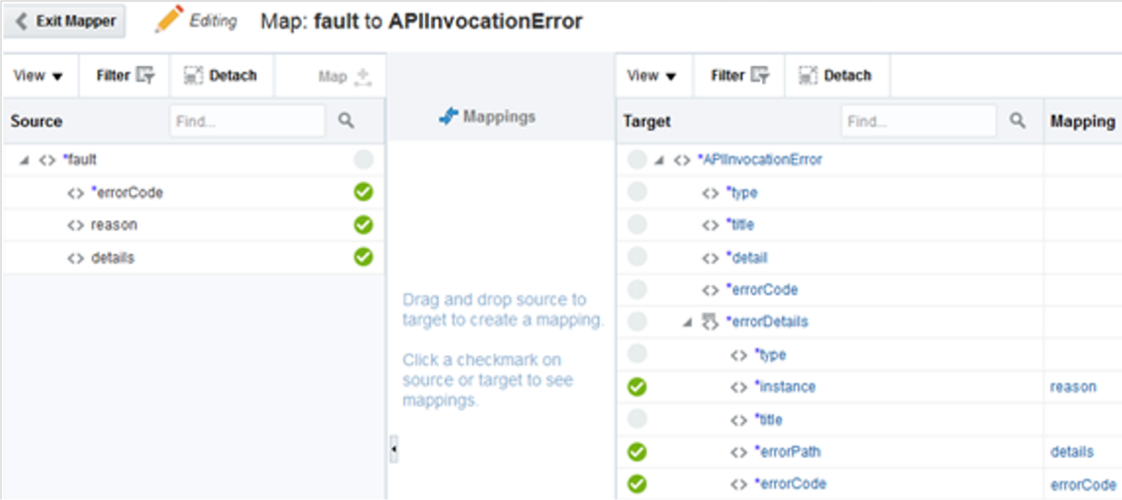
The `o:errorDetails` section is reserved for the actual cause. The prefix `o:` included is based on Oracle standards.

The top portion is used to add any integration-specific details to the fault. This is typically not necessary, but if you want to control the HTTP status, title, and details, set these values appropriately. If not entered, sufficient default values are provided by the REST Adapter.

Note:

The REST Adapter returns the downstream errors with a `500 Internal server` error code. You can override these errors and provide a custom error code by assigning an appropriate value to **APIInvocationError/errorCode** in the target mapper.

The suggested mappings to map faults raised by an outbound system to the trigger (inbound) REST Adapter are as follows:



Drag and drop source to target to create a mapping.

Click a checkmark on source or target to see mappings.

The top section is left out in this mapping and these are appropriately assigned by the REST Adapter in the previously described sample.

Unmapped faults are propagated as system faults by Oracle Integration to the inbound REST Adapter. They may not communicate the appropriate details. Therefore, it is recommended that you define the fault pipelines.

REST Service Invoked by the REST Adapter Returns a 401 Unauthorized Status Response

If a REST service invoked using the REST Adapter consistently returns a response status of `401 Unauthorized`, it may be because the application credentials configured on the Connections page are no longer valid.

The Connections page does not validate the credentials. Even if the test connection is successful, it may not be sufficient because the test connection only validates the parameters defined on the Connections page.

Because the parameters defined on the Connections page are used to call the target endpoint REST API, which is configured as part of endpoint configuration, it is strongly recommended that you test the endpoint configuration that uses this connection.

Configuration Limitation of Ten Pages in the Adapter Endpoint Configuration Wizard

Note the following issue with the REST Adapter multiple resources per endpoint use case in the Adapter Endpoint Configuration Wizard.

Symptom	Workaround	Reason
A refresh issue may occur when configuring multiple verbs and resources for the REST Adapter as a trigger connection in the Adapter Endpoint Configuration Wizard.	If the wizard does not refresh while configuring multiple operations, click **Back** to return to a previous page and then press **Next** to refresh to the current page.	The REST Adapter multiple sources per endpoint use case requires multiple iterations over the same sets of pages. This is currently a technical restriction.

Keys with Null Values During JSON Transformation are Removed

The REST Adapter removes keys with null values during JSON transformation.

For example, if the following JSON payload is sent to the REST Adapter:

```
{
"input" : "input",
"val" :  null,
"response": "response"
}
```

Oracle Integration sends the outbound request with the following JSON output.

```
{
"input" : "input",
"response": "response"
}
```

If you need the key available at the outbound service, use the following payload:

```
{
"input" : "input",
"val" :  "",
"response": "response"
}
```

Large Sample JSON File Processing with Special Characters

The sample JSON file is typically large when it has repeating structures. You can purge such repetitions because the sample only needs to represent the structure and not the instance document. However, if the JSON file is unusually large and cannot be trimmed, perform the following the steps:

1. Replace all occurrences of special characters (for example, `$`) with their corresponding codes in the sample JSON file. See JSON to XML Special Character Conversion.
2. Use the modified JSON file to complete the configuration.
3. Select the generated schema in the Adapter Endpoint Configuration Wizard.

At runtime, incoming instances of JSON documents with keys having special characters are normalized to suitable XML element names and XML documents having these elements when serialized are converted to JSON documents with special characters restored in the key names.

SSL Certification Troubleshooting Issues

For SSL certificate errors, perform the following tasks.

Topics

- Go to the **Settings** > **Certificates** tab and upload the server certificate.
- For exception errors that occur when configuring a connection with OAuth Client Credentials or OAuth Resource Owner Password Credentials:

  Carefully review the OAuth documentation and use the Custom Two-Legged security policy.
- For exception errors that occur when configuring a connection with OAuth Authorization:

  Carefully review the OAuth documentation and use the Custom Three-Legged Security Policy.

Fault and Response Pipeline Definitions in Basic Routing Integrations

You can define REST Adapter fault and response pipelines in Basic Routing integrations.

The REST Adapter on the trigger (inbound) side exposes an HTTP endpoint that HTTP clients can request for using an HTTP request, and returns an HTTP response.

If successful, the REST Adapter returns a success response. The REST Adapter returns an error response with an HTTP status belonging to the error family of codes depending on the situation. This table describes the possible cause and the REST Adapter response.

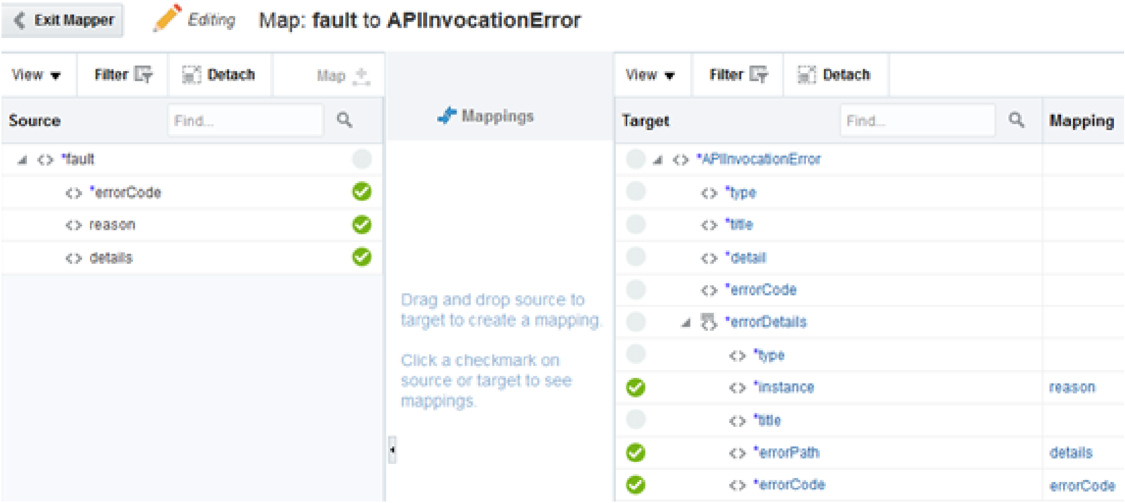
Condition	HTTP Status	Details
Invalid client request	`4xx`	There are several conditions that can cause client side failures, including: • Invalid resource URL • Incorrect query parameters • Unsupported method type • Unsupported media type • Bad data
Downstream processing errors	`5xx`	All other errors that can occur within the integration, including: • Invalid target • HTTP error response • General processing errors.

In addition, the REST Adapter also returns an error response with additional details about the error and possible steps for troubleshooting. The standard error response format is returned according to the configured response media type. The following is a sample JSON response structure:

```
{
  "type" : "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1",
  "title" : "Internal Server Error",
  "detail" : "An internal error occurred while processing the request. Please
see the fault details for the nested error details.",
  "o:errorCode" : "500",
  "o:errorDetails" : [ {
    "type" : "http://www.w3.org/Protocols/rfc2616/rfc2616-
sec10.html#sec10.4.1",
    "instance" : "{\n   \"error_message\" : \"Invalid request. Missing the
'origin' parameter.\",\n   \"routes\" : [],\n
    \"status\" : \"INVALID_REQUEST\"\n}\n",
    "title" : "Bad Request",
    "o:errorPath" : "GET http://maps.googleapis.com/maps/api/directions/json?
destination=Montreal returned a response status of
      400 Bad Request",
    "o:errorCode" : "APIInvocationError"
  } ]
}
```

The errorDetails section is reserved for the actual cause. You must configure the fault pipelines to map the target faults into this element. The top portion is used to add any integration-specific details to the fault. This is typically not necessary, but if you want to control the HTTP status, title, and details, then set these values appropriately. If not entered, sufficient default values are provided by the adapter.

The suggested mappings to map faults raised by an outbound system to the trigger (inbound) REST Adapter are as follows:



Drag and drop source to target to create a mapping.

Click a checkmark on source or target to see mappings.

The top section is left out in this mapping and so these are appropriately assigned by the adapter in the previously described sample.

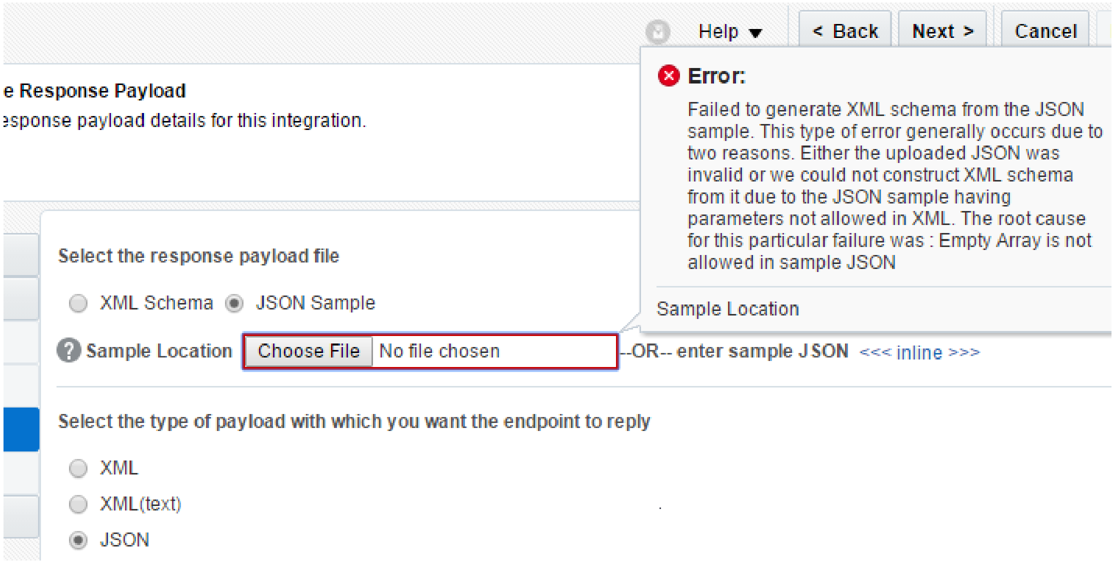
Unmapped faults are propagated as system faults by Oracle Integration to the inbound adapter. They may not communicate the appropriate details. Therefore, it is recommended that you define the fault pipelines.

Note:

Fault pipelines are only available with Basic Map Data integrations.

Empty Arrays Are Not Supported in Sample JSON Files

When configuring the REST Adapter, if a JSON property in the included JSON sample file has an empty array, you receive the following error message. Note the last part of the message. Modify the JSON sample file to include a value for the JSON property.



Error:

Failed to generate XML schema from the JSON sample. This type of error generally occurs due to two reasons. Either the uploaded JSON was invalid or we could not construct XML schema from it due to the JSON sample having parameters not allowed in XML. The root cause for this particular failure was : Empty Array is not allowed in sample JSON

Sample Location

Select the response payload file

○ XML Schema ● JSON Sample

Sample Location: Choose File No file chosen --OR-- enter sample JSON <<< inline >>>

Select the type of payload with which you want the endpoint to reply

○ XML
○ XML(text)
● JSON

Invoke Endpoint URI Must Match the Base URI + Resource URI in REST Adapter

While designing the REST Adapter in the Adapter Endpoint Configuration Wizard, carefully review the contents on the Summary page. The endpoint URI must match the invoke service URI. If you do not see the necessary values, review your invoke connection and the outbound service. The base URI in the connection and resource URI in the invoke service must add up to the endpoint URI.

JD Edwards Form Service Invocation with the REST Adapter Causes APIInvocation Error

You can receive the following error in the `icsServer-diagnostic.log` file when invoking JD Edwards Form Service from an integration in which a REST Adapter is configured as the invoke connection.

```
[2016-06-07T02:13:54.346-07:00] [icsServer] [ERROR] []
[oracle.osb.transports.jca] [tid: [ACTIVE].ExecuteThread: '14'
for queue: 'weblogic.kernel.Default (self-tuning)'] [userId: <anonymous>]
```

```
[ecid: f23c428c-9247-459c-bb9f-22cbadbeda35-0003651d,0] [APP: Service Bus JCA
Transport Provider] [oracle.soa.tracking.FlowId: 59] [FlowId:
0000LKe8vtD7MAW5Hzw0yf1NGeIK00001c] Error sending bytes to socket:
<genericRestFault><errorCode>500</errorCode><errorPath><![CDATA[POST
http://den60208jems.my_domain.com:9516/jderest/formservice returned a
response status of 500 Internal Server Error]]></errorPath><instance><!
[CDATA[{[[
"message" : "Can not deserialize instance of java.util.ArrayList out of
START_OBJECT token\n at [Source: java.io.StringReader@68f2c85d; line: 1,
column:
218] (through reference chain:
com.oracle.e1.jdemf.FormRequest[\"formInputs\"])",   "exception" :
"com.fasterxml.jackson.databind.JsonMappingException",
"timeStamp" : "2016-06-07:03.13.54" }]]></instance></genericRestFault>
]]
```

This error occurs because the REST Adapter has only one array element. JSON documents containing arrays in the REST Adapter require at least two array elements for the adapter to generate a valid XML schema. For example:

```
"formInputs": [ "input1" ]
```

cannot be handled as an array unless another cell is added in the sample JSON:

```
"formInputs": [ "input1", "input2" ]
```

REST Adapter Data is Only Saved When You Click Next

When configuring the REST Adapter in the Adapter Endpoint Configuration Wizard, you must click **Next** to save your changes and move to the next page of the wizard. For example, if you configure details on the Request page, click the tab of the Basic Info page in the left pane, then click **Next** to return to the Request page, none of your previous configurations were saved, and the page is empty.

Convert XML to a JSON Document

You can convert XML to a JSON document. Oracle Integration resolves an XML element with a number value to XML schema with a type of number, which converts the XML to a JSON document with a type of number.

For example:

- XML:

  ```
  <Phone>23249480</Phone>
  ```

- Generated XSD:

  ```
  <element name="phone" type="integer"/>
  ```

- JSON:

  ```
  "Phone": 23249480
  ```

The workaround is to use a string value for the phone number in the sample XML. The XML schema generated has a type of string. At runtime, the XML to JSON conversion produces the desired JSON. For example:

- XML:

  ```
  <Phone>a23249480</Phone> <!-- modified -->
  ```

- Generated XSD:

  ```
  <element name="phone" type="string"/>
  ```

At runtime:

- XML

  ```
  <Phone>23249480</Phone>
  ```

- JSON

  ```
  "Phone": "23249480"
  ```

Supported Special Characters in JSON Samples

The following special characters are supported in JSON samples.

- " " (blank space)
- /
- \\
- ;
- (
-)
- &
- ,
- #
- ?
- <
- >

content-type is Missing for an Asynchronous Flow

The `content-type` is missing for an asynchronous flow.

Assume you create the following integration:

1. Configure a REST Adapter connection with another Oracle Integration REST endpoint.
2. Configure a trigger REST Adapter and an invoke REST Adapter with an asynchronous flow.
3. Activate and invoke the integration.

   The `content-type` is missing.

   The `content-type` is ideally not required when the `content-length` is `0`, but `content-type text/plain` is added as the default `content-type` by some layers. Both are correct and permissible.

REST URLs Exceeding 8251 Characters Fail

The upper limit of characters that work in REST URLs in integrations with the REST Adapter is 8251. If you exceed this limit, a `414 Request-URI Too Large` error occurs.

Send a "null" Value Instead of "" for Any Specific Key in JSON Through the REST Adapter

If you want to send a `"null"` value instead of `""` for any specific key in JSON through the REST Adapter, you must map `"xsi:nil=true"` through the mapper for that specific key.

7

REST Adapter Samples

You can use the REST Adapter in end-to-end scenarios such as the following:

Topics:

Build an Integration that Exposes the REST API Using the REST Adapter

The REST Adapter can be used in scenarios such as integrating with Twitter. Twitter provides several REST endpoints for accessing resources. This use case describes how to access a protected resource from Twitter using the Basic Authentication security policy.

Obtain the Twitter Credentials

1. Obtain the necessary Twitter connection details from the Twitter developer page at https://dev.twitter.com. These keys are required for configuring the Twitter Adapter on the Connections page. See *Using the Twitter Adapter with Oracle Integration 3* for specific details.
    - Consumer key
    - Consumer secret
    - Access token
    - Access token secret

Configure the Twitter Adapter

1. In the Credentials dialog on the Connections page of Oracle Integration, complete the following fields with the information obtained from Twitter. Note that the **Custom Security Policy** security policy is displayed by default, and cannot be deselected.
    - In the **Consumer Key** field, enter the consumer key.
    - In the **Consumer Secret** field, enter the consumer secret.
    - In the **Access Token** field, enter the access token.
    - In the **Access Secret** field, enter the access token secret.

Configure the REST Adapter

1. In the Connections page of Oracle Integration, complete the following fields.
    - In the Connection Properties dialog, select **REST API Base URL** and specify the connection URL.
    - In the Credentials dialog, select **Basic Authentication** as the security policy and specify the applicable user name and password.

Create an Integration

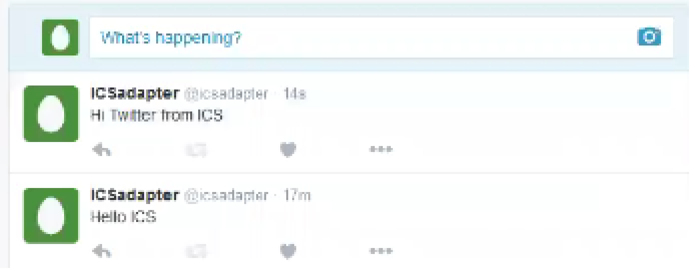
1. Drag a REST Adapter to the trigger side, and configure it as follows:
   - Specify the following parameters on the Basic Info page:
     - Select the **POST** action.
     - Select **Configure a request payload for this endpoint**.
     - Select **Configure this endpoint to receive the response**.
   - Specify the request schema on the Request page.
   - Specify the response schema on the Response page.
2. Drag a Twitter Adapter to the invoke side, and configure it as follows:
   - Select the **Tweet** operation.
3. In the request mapper, configure the appropriate source to target mapping.
4. In the response mapper, configure the appropriate source to target mapping.

Invoke the Integration

1. Invoke the integration from a browser:

   ```
   https://host:port/integration/flowapi/rest/TWEET/v01/tweet?status=Hi
   Twitter from ICS
   ```

   This posts the request status to Twitter.
2. Log in to the Twitter account.
3. Note the request message and the response message.



ICSadapter @icsadapter · 14s
Hi Twitter from ICS

ICSadapter @icsadapter · 17m
Hello ICS