Oracle® Cloud

Using the REST Adapter with Oracle Integration
# Contents

## Preface

- **Audience** vi
- **Documentation Accessibility** vi
- **Related Resources** vi
- **Conventions** vii

## 1 Understand the REST Adapter

- **REST Adapter Capabilities** 1-1
- **REST Adapter Restrictions** 1-4
- **REST Adapter Use Cases** 1-5
- **Workflow to Create and Add a REST Adapter Connection to an Integration** 1-7

## 2 REST Adapter Concepts

## 3 Create a REST Adapter Connection

- **Prerequisites for Creating a Connection** 3-1
- **Create a Connection** 3-2
  - **Add a Contact Email** 3-3
  - **Configure Connection Properties for Invoke Connections** 3-3
  - **Configure Connection Security for Invoke Connections** 3-5
  - **Configure an Agent Group** 3-9
  - **Test the Connection** 3-10
- **Upload an SSL Certificate** 3-10

## 4 Add the REST Adapter Connection to an Integration

- **Add the REST Adapter as a Trigger Connection** 4-1
  - **REST Adapter Trigger Basic Information Page** 4-2
  - **REST Adapter Trigger Request Parameters Page** 4-4
  - **REST Adapter Trigger Request Page** 4-4
Implement Common Patterns Using the REST Adapter

- Configure the REST Adapter to Consume a REST API Protected with 2-Legged OAuth Token-Based Authentication 5-2
- Configure the REST Adapter to Consume a REST API Protected with 3-Legged OAuth Token-Based Authentication 5-7
- Understand Security Configurations for Invoking Popular OAuth-Protected APIs 5-11
- Configure the REST Adapter to Consume a REST API Protected with the API Key 5-12
- Configure the REST Adapter to Consume a REST API Protected with OAuth 1.0a One-Legged Authentication 5-13
- Allow Client Applications to Consume an Integration Exposed as an OAuth-Protected REST API 5-15
- Override the Endpoint URI/Host Name for an External REST API at Runtime 5-19
- Map to Construct the Payload for an External REST API that Accepts multipart/form-data as the Content Type 5-20
- Configure the REST Adapter to Consume an External REST API Described Using a Swagger Document 5-25
- Configure the REST Adapter to Consume an External REST API Described Using a RAML Document 5-25
- Configure the REST Adapter to Consume an External REST API with No Metadata Described in a Document 5-26
- 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 5-28
- Map JSON when the REST Adapter Request is Configured with multipart/form-data 5-29
- 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 5-30
- Configure the REST Adapter to Expose an Integration as a REST API 5-34
Configure a REST Adapter to Consume a REST API that Expects Custom HTTP Header Properties 5-35
Configure the REST Adapter to Consume an Amazon Web Services (AWS) REST API 5-36
Enter q as a Standard HTTP Query Parameter with the Query as a Value 5-37
JSON to XML Special Character Conversion 5-37

6 Troubleshoot the REST Adapter

REST Services that Return Multiple Successful Responses 6-1
Error Handling with the REST Adapter 6-2
REST Service Invoked by the REST Adapter Returns a 401 Unauthorized Status Response 6-4
Configuration Limitation of Ten Pages in the Adapter Endpoint Configuration Wizard 6-5
Keys with Null Values During JSON Transformation are Removed 6-5
Large Sample JSON File Processing with Special Characters 6-6
SSL Certification Troubleshooting Issues 6-6
Fault and Response Pipeline Definitions in Basic Routing Integrations 6-6
Empty Arrays Are Not Supported in Sample JSON Files 6-8
Invoke Endpoint URI Must Match the Base URI + Resource URI in REST Adapter 6-9
JD Edwards Form Service Invocation with the REST Adapter Causes APIInvocation Error 6-9
REST Adapter Data is Only Saved When You Click Next 6-10
Convert XML to a JSON Document 6-10
Supported Special Characters in JSON Samples 6-11
content-type is Missing for an Asynchronous Flow 6-11
REST URLs Exceeding 8251 Characters Fail 6-12

7 REST Adapter Samples

Build an Integration that Exposes the REST API Using the REST Adapter 7-1
Preface

This guide describes how to configure the REST Adapter as a connection in an integration in Oracle Integration.

**Note:**
The information in this guide applies to all of your Oracle Integration instances. It doesn’t matter which edition you’re using, what features you have, or who manages your cloud environment. You’ll find what you need here, including notes about any differences between the various flavors of Oracle Integration when necessary.

**Topics**
- Audience
- Documentation Accessibility
- Related Resources
- Conventions

**Audience**

This guide is intended for developers who want to use the REST Adapter in integrations in Oracle Integration.

**Documentation Accessibility**


**Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit [http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info](http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info) or visit [http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs](http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs) if you are hearing impaired.

**Related Resources**

See these Oracle resources:
- Oracle Cloud
  http://cloud.oracle.com
- Using Integrations in Oracle Integration
- Using the Oracle Mapper with Oracle Integration

Conventions

The following text conventions are used in this document:

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

REST Adapter Capabilities When Exposing an Integration as a REST API by Configuring the Connection as a Trigger

- Support for uploading complex XML schema definitions as a zipped archive to define data definitions for XML content during REST Adapter configuration. See Complex Schema Support.
- Support for uploading 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.

- Supports configuration of the following:
  - Relative resource URI.
  - Support for HTTP methods GET, PUT, POST, DELETE, and PATCH.
  - Template and query parameters.
  - Support for a request/response payload.
    * Support for JSON, XML, and URL-form-encoded payloads.
    * Support for homogenous JSON arrays including top-level arrays.
    * Support for 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.

- REST APIs exposed using the REST Adapter are protected using Basic Authentication and OAuth token-based authentication.

See Configuration Parameters.

- Enforces inbound and message and attachment size limitations:
  - Ensures that incoming (trigger) message requests without attachments do not exceed 10 MB in size. 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 10 MB, an HTTP error code message is returned to the client: 413 Request entity too large.
  - Ensures that incoming (trigger) JSON attachments do not exceed 1 GB in size. If the size of the JSON attachment exceeds 1 GB, an HTTP error code message is returned to the client: 413 Request entity too large.
  - 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 10 MB in size. If the size of the structured message exceeds 10 MB, an HTTP error code message is returned to the client: 413 Request entity too large.

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

- Support for 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).

- REST APIs exposed using the REST Adapter can be configured to be CORS-compliant (see Cross-Origin Resource Sharing (CORS)).

- Support for exposing a REST endpoint that can accept the request and process it asynchronously.
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 (see View the Metadata for the Inbound REST Endpoint in Swagger Format).

REST Adapter Capabilities When Consuming External REST APIs by Configuring the Connection as an Invoke

- Enforces outgoing message and attachment size limitations:
  - 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/form-data and binary/octet-stream) from a client do not exceed 1 GB in size.
  - Ensures that outgoing (invoke) messages returning structured message payloads (any content-type header containing JSON, XML, HTML, YAML, or YML) from a client do not exceed 10 MB in size.

- Support for consuming any REST API described using Swagger 2.0/RAML 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 (see Swagger and RAML Document Support for Describing External REST APIs).

- Supports configuration of the following (see Configuration Parameters):
  - Relative resource URI.
  - Support for HTTP methods GET, PUT, POST, DELETE, and PATCH.
  - Template and query parameters.
  - Support for a request/response payload:
    * Support for JSON, XML, Raw (inline and unstructured), and URL-form-encoded payloads.
    * Support for homogenous JSON arrays.
    * Support for multidimensional JSON arrays (see Homogenous Multidimensional Array Support in JSON Documents).
    * Delivery of form parameters as part of a request body.

- Support for uploading 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.

- Support for uploading sample JSON documents to define data definitions during REST Adapter configuration.
• Support for uploading complex XML schema definitions as a zipped archive to define data definitions for XML content during REST Adapter configuration (see Complex Schema Support).

• Support for accessing and setting standard and custom HTTP headers exposed by external REST APIs (see Standard and Custom Header Support).

• Support for 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 for consuming external REST APIs that are not described using Swagger/RAML 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.

• Support for consuming external REST APIs that are protected using transport level security.

• Support for 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, and Amazon Web Services (AWS) Signature Version 4. There is also support for consuming APIs that are unprotected.

• Support for invoking 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.

• Extensibility support to access plurality of OAuth 2 providers (see Extensibility Support for Multiple OAuth Providers).

• Support for dynamically changing the (invoke) outbound endpoint configuration (see Support for Dynamic REST Endpoints).

• Support for 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.

External REST APIs hosted on a two-way SSL server require client side (Oracle Integration) identity. Oracle Integration provides support for exchanging the client side identity with the server hosting the external API.

REST Adapter Restrictions

Note the following REST Adapter restrictions.

• The OAuth Authorization Code Credentials security policy does not currently work with Microsoft endpoints with the REST Adapter. This is because this security
policy sends an access token in the query parameter and Microsoft endpoints need the access token in the authorization header. As a workaround, use the OAuth Custom Three Legged Flow security policy.

- 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 described using OpenAPI 3.0 are not supported.
- Consuming external REST APIs that are protected using NTLM or digest token-based authentication are not supported.
- Consumption of Swagger documents of external REST APIs that have metadata regarding content types such as multipart/form-data, multipart/mixed, and application/octet-stream are not supported.
- 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 \( a+b \) before sending.

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*. 
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.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>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.</td>
<td>Create a REST Adapter Connection</td>
</tr>
<tr>
<td>2</td>
<td>Create the integration. When you do this, you add trigger and invoke connections to the integration.</td>
<td>Create Integrations and Add the REST Adapter Connection to an Integration</td>
</tr>
<tr>
<td>3</td>
<td>Map data between the trigger connection data structure and the invoke connection data structure.</td>
<td>Map Data of Using Integrations in Oracle Integration</td>
</tr>
<tr>
<td>4</td>
<td>(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).</td>
<td>Manage Lookups of Using Integrations in Oracle Integration</td>
</tr>
<tr>
<td>5</td>
<td>Activate the integration.</td>
<td>Manage Integrations of Using Integrations in Oracle Integration</td>
</tr>
<tr>
<td>6</td>
<td>Monitor the integration on the dashboard.</td>
<td>Monitor Integrations of Using Integrations in Oracle Integration</td>
</tr>
<tr>
<td>7</td>
<td>Track payload fields in messages during runtime.</td>
<td>Assign Business Identifiers for Tracking Fields in Messages and Manage Business Identifiers for Tracking Fields in Messages of Using Integrations in Oracle Integration</td>
</tr>
<tr>
<td>8</td>
<td>Manage errors at the integration level, connection level, or specific integration instance level.</td>
<td>Manage Errors of Using Integrations in Oracle Integration</td>
</tr>
</tbody>
</table>
The following sections describe REST Adapter capabilities in more detail.

Topics:

• Configuration Parameters
• Standard and Custom Header Support
• Authentication Types
• Extensibility Support for Multiple OAuth Providers
• Role-Based Connections
• Cross-Origin Resource Sharing (CORS)
• Swagger and RAML Document Support for Describing External REST APIs
• Homogenous Multidimensional Array Support in JSON Documents
• Multipart Attachment Support for Trigger and Invoke Connections
• On-Premises REST API Support with the Agent
• View the Metadata for the Inbound REST Endpoint in Swagger Format
• RFC 3986 Support for Encoding Query Parameters
• Support for application/octet-stream MIME Attachment (Raw) Payloads
• Security is Not Required for Swagger Definition and Metadata Catalog URL Connections
• REST Endpoint Metadata and a Swagger Link to a REST Metadata Description
• Support for Dynamic REST Endpoints
• Complex Schema Support
• Nonstandard JWT Token Support

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

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 Create an Orchestrated 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 comma val2 comma 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 cannot store multiple headers with the same name. The WSDL can only store one element with one unique name.

Authentication Types

The REST Adapter supports the invocation of external REST endpoints supporting the following types of authentication:

• 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 1.0a One Legged Authentication
• Amazon Web Services (AWS) Signature Version 4

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

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 for Invoke Connections 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.

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. Basic authentication is the only security policy available. 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.

**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 inbound direction.

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

- 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
<html>
<script language="javascript">

Chapter 2

2-4
```
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
```

Swagger and RAML Document Support for Describing External REST APIs

The REST Adapter provides support for consuming REST APIs that are described in either a Swagger or RAML document.
• RESTful API Modeling Language (RAML): A language for describing RESTful APIs. RAML provides the information necessary to describe RESTful or practically-RESTful APIs (APIs that do not obey all REST constraints).

• Swagger: A specification for describing, producing, consuming, and visualizing RESTful web services.

The following example shows a Swagger 2.0 file. This file contains two main resources:

• /Book. This resource contains get and post methods and /Book/{id}, /Book/hello, and /Book/search subresources.

• /Author. This resource contains a get method and an /Author/{id} subresource.

When configuring an invoke (outbound) REST Adapter in the Adapter Endpoint Configuration Wizard, the resources and subresources are displayed for selection as business objects and the methods are displayed for selection as operations to perform on the business objects.

When creating the REST Adapter connection, you select Swagger Definition URL in the Connection Type field and specify the URL in the Connection URL field of the Connection Properties dialog.

```json
{
    "swagger" : "2.0",
    "info" : {
        "version" : "1.0",
        "title" : "RestServiceForBooks"
    },
    "host" : "host_name:8080",
    "basePath" : "/Test/rest",
    "schemes" : ["http"],
    "paths" : {
        "/Book" : {
            "get" : {
                "operationId" : "getBooks",
                "description" : "Returns all the available books in the store",
                "produces" : ["application/xml", "application/json"],
                "responses" : {
                    "default" : {
                        "schema" : {
                            "$ref" : "#/definitions/Books"
                        }
                    }
                }
            },
            "post" : {
                "operationId" : "postBook",
                "description" : "Creates a new book item",
                "produces" : ["application/xml", "application/json"],
                "consumes" : ["application/xml", "application/json"],
                "parameters" : [
```
"name" : "Book",
"in" : "body",
"required" : true,
"schema" : { "$ref" : "#/definitions/Book" }
],
"responses" : {
"default" : {
"schema" : { "$ref" : "#/definitions/Book" }
}
}
},
"/Book/{id}" : {
"get" : {
"operationId" : "getSingleBook",
"description" : "Returns a book with specific id",
"produces" : [ "application/xml", "application/json" ],
"parameters" : [
{
"name": "id",
"in": "path",
"required" : true,
"type" : "string"
}
],
"responses" : {
"default" : {
"schema" : { "$ref" : "#/definitions/Book" }
}
}
}
}
,"/Book/hello" : {
"get" : {
"operationId" : "sayHelloToBook",
"description" : "says hello to a book",
"produces" : [ "application/xml", "application/json" ],
"responses" : {
"default" : {
"schema" : { "type" : "string" }
}
}
}
}
,"/Book/search" : {
"get" : {
"operationId" : "searchBook",
"description" : "Returns a list of books that match query param",
"produces" : [ "application/xml", "application/json" ],
"parameters" : [
{
"name": "name",
"in": "query",
"required" : false,
"type": "string"
}
,"responses": {
  "default": {
    "schema": {
      "$ref": "#/definitions/Books"
    }
  }
}
,"/Author": {
  "get": {
    "operationId": "getAuthors",
    "description": "Returns a list of authors",
    "produces": [
      "application/xml",
      "application/json"
    ],
    "responses": {
      "default": {
        "schema": {
          "$ref": "#/definitions/Authors"
        }
      }
    }
  }
},
="/Author/{id}\": {
  "get": {
    "operationId": "getSingleAuthor",
    "description": "Returns a Author with specific id",
    "produces": [ "application/xml", "application/json" ],
    "parameters": [
      {"name": "id", "in": "path", "required": true, "type": "string"}
    ],
    "responses": {
      "default": {
        "schema": { "$ref": "#/definitions/Author" }
      }
    }
  }
},
"definitions": {
  "Author": {
    "type": "object",
    "properties": {
      "id": { "type": "string" }
    }
  }
}
The following example shows a RAML file. The file contains the schemas that use the service. This file contains two main resources:

- /Author. This resource contains a get method and an /Author/{id} subresource.
- /Book. This resource contains get and post methods and /Book/{id} and /Book/search subresources.

When configuring an invoke (outbound) REST Adapter in the Adapter Endpoint Configuration Wizard, the resources and subresources are displayed for selection as business objects and the methods are displayed for selection as operations to perform on the business objects.

When creating your REST Adapter connection, you select RAML Definition URL in the Connection Type field and specify the URL in the Connection URL field of the Connection Properties dialog.

```yaml
#%RAML 0.8
title: API for Books
version: v1
baseUri: "http://host_name:8080/Test/rest"
protocols: [ HTTP ]
schemas:
  - authors-jsonschema: |
    
    "$schema" : "http://json-schema.org/draft-03/schema",
    "type": "object",
    "properties": {
      "id": {
        "type": "string"
      },
      "firstName": {
        "type": "string"
      },
      "lastName": {
        "type": "string"
      }
    },
    "required": [
      "id",
      "firstName",
      "lastName"
    ]
```

---

Oracle
- author-jsonschema: |
  |
  | "$schema":"http://json-schema.org/draft-03/schema",
  |
  | "type":"object",
  | "properties":{
  |   "id":{
  |     "type":"string"
  |   },
  |   "firstName":{
  |     "type":"string"
  |   },
  |   "lastName":{
  |     "type":"string"
  |   }
  | },
  | "required":[
  |   "id",
  |   "firstName",
  |   "lastName"
  | ]
  |
- books-jsonschema: |
  |
  | "$schema":"http://json-schema.org/draft-03/schema",
  |
  | "type":"object",
  | "properties":{
  |   "items":{
  |     "type":"array",
  |     "items":{
  |       "type":"object",
  |       "properties":{
  |         "id":{
  |           "type":"string"
  |         },
  |         "name":{
  |           "type":"string"
  |         },
  |         "ISBN":{
  |           "type":"string"
  |         },
  |         "price":{
  |           "type":"integer"
  |         },
  |         "author":{
  |           "type":"array",
  |           "items":{
  |             "type":"object",
  |             "properties":{
  |               "id":{
  |                 "type":"string"
  |               },
  |               "firstName":{
  |                 "type":"string"
  |               },
  |               "lastName":{
  |                 "type":"string"
  |               }
  |             }
  |         }
  |       }
  |   }
  | }
  |}
"id":{
   "type":"string"
},
"firstName":{
   "type":"string"
},
"lastName":{
   "type":"string"
},
"required":[
   "id",
   "firstName",
   "lastName"
]
},
"publisher":{
   "type":"object",
   "properties":{
      "id":{
         "type":"string"
      },
      "name":{
         "type":"string"
      },
      "location":{
         "type":"string"
      }
   },
   "required":[
      "id",
      "name",
      "location"
   ]
},
"required":[
   "id",
   "name",
   "ISBN",
   "price",
   "author",
   "publisher"
]
}
}

- book-jsonschema: |

   "$schema":"http://json-schema.org/draft-03/schema",
   "type":"object",
   "properties":{
"id":{
   "type":"string"
},
"name":{
   "type":"string"
},
"ISBN":{
   "type":"string"
},
"price":{
   "type":"integer"
},
"author":{
   "type":"array",
   "items":{
      "type":"object",
      "properties":{
         "id":{
            "type":"string"
         },
         "firstName":{
            "type":"string"
         },
         "lastName":{
            "type":"string"
         }
      },
      "required":[
         "id",
         "firstName",
         "lastName"
      ]
   }
},
"publisher":{
   "type":"object",
   "properties":{
      "id":{
         "type":"string"
      },
      "name":{
         "type":"string"
      },
      "location":{
         "type":"string"
      }
   },
   "required":[
      "id",
      "name",
      "location"
   ]
},
"required":[]
"id",
"name",
"ISBN",
"price",
"author",
"publisher"
]
}

/Author:
  get:
    responses:
      200:
        body:
          application/xml:
            schema: authors-jsonschema
            example: |
              <?xml version="1.0" encoding="UTF-8"?>
              <authors></authors>
          application/json:
            schema: authors-jsonschema
            example: |
            {           "authors" : ""
            }

  /{id}:
    get:
      responses:
        200:
          body:
            application/xml:
              schema: author-jsonschema
              example: |
              <?xml version="1.0" encoding="UTF-8"?>
              <author /></author>
            application/json:
              schema: author-jsonschema
              example: |
              {           "author" : ""
              }

/Book:
  post:
    body:
      application/xml:
        schema: book-jsonschema
      application/json:
        schema: book-jsonschema
    responses:
      200:
        body:
          application/xml:
            schema: book-jsonschema
            example: |
            <?xml version="1.0" encoding="UTF-8"?>
            <book>

application/json:
  schema: book-jsonschema
  example: |
  {
    "book" : {
      "price" : ""
  }
}

get:
  responses:
  200:
    body:
      application/xml:
        schema: books-jsonschema
        example: |
          <?xml version="1.0" encoding="UTF-8"?>
          <book>
            <price></price>
          </book>
      application/json:
        schema: books-jsonschema
        example: |
          {
            "book" : {
              "price" : ""
            }
          }

/search:
  get:
    queryParameters:
    name:
      responses:
      200:
        body:
          application/xml:
            schema: books-jsonschema
            example: |
              <?xml version="1.0" encoding="UTF-8"?>
              <book>
                <price></price>
              </book>
          application/json:
            schema: books-jsonschema
            example: |
              {
                "book" : {
                  "price" : ""
                }
              }

/{id}:
  get:
    responses:
    200:
body:
  application/xml:
    schema: book-jsonschema
    example: |
      <?xml version="1.0" encoding="UTF-8"?>
      <book>
        <price></price>
      </book>
  application/json:
    schema: book-jsonschema
    example: |
      {
        "book" : {
          "price" : ""
        }
      }

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 "recordsData").

{  
  "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">
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.

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-wraper 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.

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

Oracle Integration provides an agent framework that enables 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 as follows:

- Select **REST API Base URL** or **Swagger Definition URL** from the *Connection Type* list and enter the appropriate URL in the *Connection URL* field of the *Connection Properties* dialog. No other connection types are supported.
- Select **Basic Authentication** or **No Security Policy** from the *Security Policy* list of the *Credentials* dialog. No other security policies are supported.
- Select the previously-created agent group in the *Select an Agent Group* dialog.

For conceptual information about the on-premises agent, see About Agents and Integrations Between On-Premises Applications and 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.

View the Metadata for the Inbound REST Endpoint in Swagger Format

You can view the metadata of an activated REST integration and then append `/swagger` to the metadata URL to view the Swagger format for the integration. The inbound REST integration can then be exposed as a Swagger connection.

1. On the Integrations page, find the integration whose endpoint URL you want to use.
2. Click the **Details** icon at the far right.
3. Click the **Endpoint URL** value (for example, `http://myPODname:7002/integration/flowapi/rest/GET_ONE_BOOK/v01/metadata`).
4. Append `/swagger` to the end of the URL, and press **Enter**.

Appending `/swagger` to the URL generates a Swagger document for the inbound integration. This URL can also be used to create a new Swagger connection in the Connection Properties dialog. You enter the Swagger URL in the *Connection URL* field and select **Swagger Definition URL** from the *Connection Type* field.
RFC 3986 Support for 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.

Support for application/octet-stream MIME Attachment (Raw) 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 Raw 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 raw 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 raw payloads can be mapped.
Security is Not Required for Swagger Definition and Metadata Catalog URL Connections

Upon activation of an integration with a REST Adapter used as a trigger, a metadata link is produced with documentation that describes the Oracle Integration REST endpoint. A Swagger description is also produced so other APIs can also consume the Oracle Integration REST endpoint. The Swagger and metadata artifacts that correspond to the Oracle Integration REST endpoint are unprotected.

The Oracle Integration REST endpoints are still protected, just the Swagger and metadata artifacts have been made unprotected to enable better discovery from third party APIs.

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.

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.

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 than the original URL, you may also have to provide a mapping for the authorization standard header.

**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">
    targetNamespace="http://xmlns.oracle.com/apps/scm/productModel/items/itemServiceV2/"/>
  <xs:complexType name="author">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
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
MTgwNzE5NTY1NToxQkhzQlpaSXM0a21BV3NhVBIc1JOTFM4OGFxU09jNlRTdmFksmczLVBqVH1WRF JwbWyxOFhmcnN6S0N6c3Fzb1JKbEh6U2IwSTdfV1VuZWFXYjVmemhJNTJ1YVn6FdBDbTBG

- 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":"MTgwNzE5NTY1NToxQkhzQlpaSXM0a21BV3NhVBIc1JOTFM4OGFxU09jNlRTdmFksmczLVBqVH1WRF JwbWyxOFhmcnN6S0N6c3Fzb1JKbEh6U2IwSTdfV1VuZWFXYjVmemhJNTJ1YVn6FdBDbTBG"
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
• Create a Connection
• Upload an SSL Certificate

Prerequisites for Creating a Connection

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

• 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.

• For SSL endpoints, obtain and upload a server certificate. For more information, see Upload an SSL Certificate.

• To create an integration that consumes external REST APIs hosted on a two-way, SSL-enabled server, satisfy the following prerequisites:
  – Ensure that the server on which the external REST APIs are hosted is enabled for two-way SSL support.
  – File a service request with Oracle Support Services to obtain the keystore file required for establishing an Oracle Integration identity to facilitate a two-way SSL-based integration.
– Import the necessary keystore file and trust certificate.

• 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.

Create a Connection

The first step in creating an integration is to create the connections to the applications with which you want to share data.

1. In the navigation pane, click Integrations, then click Connections.
2. Click Create.

Note:

You can also create a connection in the integration canvas of:

• An orchestrated integration (See Define Inbound Triggers and Outbound Invokes.)
• A basic routing integration (See Add a Trigger (Source) Connection.)

The Create Connection — Select Adapter dialog is displayed.

3. Select an adapter from the dialog. You can also search for the type of adapter to use by entering a partial or full name in the Search field, and clicking Search.

The Create New Connection dialog is displayed.

4. Enter the information to describe the connection.

• Enter a meaningful name to help others find your connection when they begin to create their own integrations. The name you enter is automatically added in capital letters to the Identifier field. If you modify the identifier name, do not include a blank space (for example, Sales Opportunity).

• Select the role (direction) in which to use this connection (trigger, invoke, or both). Only the roles supported by this adapter are displayed for selection. 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, then try to drag the adapter into the section you did not select, you receive an error (for example, configure an Oracle Service Cloud (RightNow) Adapter as only an invoke, but drag the adapter to the trigger section).

• Enter an optional description of the connection.
5. Click **Create**.

Your connection is created and you are now ready to configure connection details, such as email contact, connection properties, security policies, connection login credentials, and (for certain connections) agent group.

### Add a Contact Email

You can add an optional contact email address for notifications.

1. In the **Email Address** field, enter an optional email address. You do not receive automatic notifications at this address.

2. In the upper right corner, click **Save**.

### Configure Connection Properties for Invoke Connections

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

1. Click **Configure Connectivity**.

   The Connection Properties dialog is displayed.

2. From the **Connection Type** list, select the type to use:

   The swagger, RAML, and metadata catalogs are commonly used, language agnostic standards to define the capabilities of a service. The REST Adapter can parse these resource definitions, discover resources, and understand how to interact with these resources with a minimal amount of user intervention. If the target API does not define a resource model in one of these formats, select the **REST API Base URL** as the connection type, specify the base URL of the service, and model the request and the expected response using the Adapter Endpoint Configuration Wizard.
3. From the **TLS Version** list, optionally specify the Transport Layer Security (TLS) version of the target server. The TLS protocol provides privacy and data integrity between two communicating computer applications. If no version is selected, the REST Adapter uses TLSv1 by default. The selected version is used for SSL/TLS negotiation and SSL handshake in all outbound invocations of the REST API. Existing integrations and connections are not impacted.

- **TLSv1**
- **TLSv1.1**
- **TLSv1.2**

4. In the **Connection URL** field, specify the endpoint URL to use based on your selection in Step 2. The connection URL can be both HTTP and HTTPS.

<table>
<thead>
<tr>
<th>Type</th>
<th>Endpoint Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST API Base URL</td>
<td><a href="https://hostname:port/ic/api/integration/v1/flows/rest/INTEGRATION_NAME/v01/">https://hostname:port/ic/api/integration/v1/flows/rest/INTEGRATION_NAME/v01/</a></td>
</tr>
<tr>
<td>Metadata Catalog URL</td>
<td><a href="https://hostname:port/Test/mdcatalogmain.json">https://hostname:port/Test/mdcatalogmain.json</a></td>
</tr>
<tr>
<td>Swagger Definition URL</td>
<td><a href="https://hostname:port/Test/application.json">https://hostname:port/Test/application.json</a></td>
</tr>
<tr>
<td>RAML Definition URL</td>
<td><a href="https://hostname:port/Test/fullapi2.raml">https://hostname:port/Test/fullapi2.raml</a></td>
</tr>
</tbody>
</table>

5. If you are configuring the REST Adapter for use with a two-way SSL-enabled server, enter information in the following fields.

a. In the **Enable two way SSL for outbound connections** field, select **Yes**.

b. In the **Identity keystore alias name** field, enter the key alias name from the keystore file that you specified when importing the identity certificate.

6. Click **OK**.

7. Configure connection security.
Configure Connection Security for Invoke Connections

Configure security for your REST Adapter connection by selecting the security policy and security token.

1. Click **Configure Credentials**.

2. Select the security policy to use. Based on your selection, the page is referenced to display various login credential fields. You must already have created your client application to complete the necessary fields.

![Note:](#)

**Note:**

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.

<table>
<thead>
<tr>
<th>Selected Security Policy</th>
<th>Fields</th>
</tr>
</thead>
</table>
| AWS Signature Version 4  | • **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. |
<table>
<thead>
<tr>
<th>Selected Security Policy</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth Client Credentials</td>
<td>• <strong>Access Token URI</strong> — The URL from which to obtain the access token.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Client Id</strong> — The client identifier issued to the client during the registration process.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Client Secret</strong> — The client secret.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Confirm Client Secret</strong> — Reenter the client secret.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Scope</strong> — 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.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Auth Request Media Type</strong> — 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.</td>
</tr>
<tr>
<td>OAuth Resource Owner Password Credentials</td>
<td>• <strong>Access Token URI</strong> — The URL from which to obtain the access token.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Client Id</strong> — The client identifier issued to the client during the registration process.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Client Secret</strong> — The client secret.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Confirm Client Secret</strong> — Reenter the client secret.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Scope</strong> — 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.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Auth Request Media Type</strong> — The format of the data you want to receive.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Username</strong> — The resource owner’s user name.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Password</strong> — The resource owner’s password.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Confirm Password</strong> — Reenter the password.</td>
</tr>
<tr>
<td>OAuth Authorization Code Credentials</td>
<td>• <strong>Client Id</strong> — The client identifier issued to the client during the registration process.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Client Secret</strong> — The client secret.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Confirm Client Secret</strong> — Reenter the client secret.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Authorization Code URI</strong> — The URI from which to request the authorization code.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Access Token URI</strong> — URI to use for the access token.</td>
</tr>
<tr>
<td>Note: The OAuth Authorization Code</td>
<td>• <strong>Scope</strong> — 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.</td>
</tr>
<tr>
<td>Credentials does not currently work with</td>
<td>REST Adapter Restrictions.</td>
</tr>
<tr>
<td>Microsoft endpoints.</td>
<td></td>
</tr>
</tbody>
</table>

The OAuth Authorization Code Credentials security policy does not currently work with Microsoft endpoints. See REST Adapter Restrictions.
<table>
<thead>
<tr>
<th>Selected Security Policy</th>
<th>Fields</th>
</tr>
</thead>
</table>
| OAuth Custom Three Legged Flow | • **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.  
  • **Access Token Request** — The access token request to use to fetch the access token. Specify the request using CURL syntax. For example:  
    ```bash
    -X POST -H headers -d string_data
    access_token_uri?query_parameters
    ```  
  • **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:  
    ```bash
    -X POST -H headers -d string_data
    access_token_uri?query_parameters
    ```  
  • **Sauth_code** — Use regex to identify the authorization code.  
  • **Saccess_token** — Use a regular expression (regex) to retrieve the access token.  
  • **Srefresh_token** — Use regex to retrieve the refresh token.  
  • **Sexpiry** — Use regex to identify when the access token expires.  
  • **Stoken_type** — Use regex to identify the access token type.  
  • **access_token_usage** — Specify how to pass the access token to access a protected resource. You can pass the token as a bearer token or as a query parameter. For example:  
    ```bash
    -H Authorization: Bearer ${access_token}
    ``` |
### Selected Security Policy

**OAuth Custom Two Legged Flow**

See [Configure the REST Adapter to Consume a REST API Protected with 2-Legged OAuth 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:

```bash
-X POST method -H headers -d string_data access_token_uri?query_parameters
```

- **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:

```bash
-X POST method -H headers -d string_data access_token_uri?query_parameters
```

- **access_token** — Use regex to identify the access token.
- **refresh_token** — Use regex to identify the refresh token.
- **expiry** — Use regex to identify when the access token expires.
- **token_type** — Use regex to identify the access token type.
- **access_token_usage** — Specify how to pass the access token to access a protected resource. You can pass the token as a bearer token or as a query parameter. For example:

```
-H Authorization: Bearer ${access_token}
```

---

**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:

```
?api_key=${access_token}
```

For example:

```bash
http://someapi.com/employee?
api_key=ASDFADAX
```

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

<table>
<thead>
<tr>
<th>Security Policy</th>
<th>Fields</th>
</tr>
</thead>
</table>
| 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.  |
| No Security Policy                                   | If you select this security policy, no additional fields are displayed. |

3. Click **OK**.

**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. See **Understand Security Configurations for Invoking Popular OAuth-Protected APIs**.

4. Test the connection.

**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.

### Configure an Agent Group

Configure an agent group for accessing the service hosted on your premises behind the fire wall.

1. Click **Configure Agents**.

   The Select an Agent Group page appears.

2. Click the name of the agent group.

3. Click **Use**.

   To configure an agent group, you must download and install the on-premises connectivity agent. See **Download and Run the On-Premises Agent Installer**.
Test the Connection

Test your connection to ensure that it is successfully configured.

1. In the upper right corner of the page, click Test.

2. If your adapter connection uses a WSDL, you are prompted 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.

   If successful, the following message is displayed and the progress indicator shows 100%.
   
   Connection *connection_name* was tested successfully.

3. If your connection was unsuccessful, an error message is displayed with details. Verify that the configuration details you entered are correct.

4. When complete, click Save, then click Close.

Upload an SSL Certificate

Certificates are used to validate outbound SSL connections. If you make an SSL connection in which the root certificate does not exist in Oracle Integration, an exception 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.

To upload an SSL certificate:

1. In the navigation pane, click Integrations, then click the < arrow next to Designer.

2. Click Settings > Certificates.

   All certificates currently uploaded to the trust store are displayed in the Certificates dialog. The Filter By > Type list displays the following details:
   - **Preinstalled**: Displays the certificates automatically installed in Oracle Integration. These certificates cannot be deleted.
   - **Uploaded**: Displays the certificates uploaded by individual users. These certificates can be deleted and updated.

   You can also search for certificates in the Search field. The search results are limited to a maximum of ten records sorted by name for performance and usability reasons. To ensure that your search results are more granular, enter as much of the certificate name as possible.

3. Click Upload at the top of the page.
4. In the Upload Certificate dialog box, select the certificate type. Each certificate type enables Oracle Integration to connect with external services.

- **Trust Certificate**: Use this option to upload a trust certificate.
  
  a. Enter a unique alias for the certificate.
  
  b. Click Browse, then select the trust file (for example, .cer or .crt) to upload.

- **Message Protection Certificate**: Use this option to upload a keystore certificate with SAML token support. Create, read, update, and delete (CRUD) operations are supported on this type of certificate.
  
  a. Enter a unique alias for the certificate.
  
  b. Click Browse, then select the certificate file (.cer or .crt) to upload.

- **Identity Certificate**: Use this option to upload a certificate for two-way SSL communication.
  
  a. Click Browse, then select the keystore file (.jks) to upload.
  
  b. Enter the password of the keystore being imported.
  
  c. Enter the comma-separated list of aliases from the keystore being imported.
  
  d. Enter the comma-separated list of passwords corresponding to key aliases.
  
  e. If you want to display the passwords in clear text, select **Show Key Password(s)**. This enables you to ensure that you are correctly entering a list of keystore passwords.

5. Click **Upload**.

6. Click the certificate name to view details such as the subject of the certificate, the issuer of the certificate, the date the certificate was issued, and the date the certificate expires.
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.

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 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 Operation Selection Page
- Summary Page
## REST Adapter Trigger 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.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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/ic/api/integration/v1/flows/rest/INTEGRATION_NAME/VERSION
  ```
  For example, if the integration name is `ExposeFlowAsRESTResource`, the URL becomes:  
  ```
  http://host:port/ic/api/integration/v1/flows/rest/EXPOSEFLOWASRESTRESOURCE
  ```
  You can override the URL, except for the fixed part at the beginning:  
  ```
  host:port/ic
  ```
### What action does the endpoint perform?

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.

### Based on your selections, you can add parameters or configure a request and/or response for this endpoint

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.

REST Adapter Trigger Request Parameters Page

Enter the REST Adapter request parameters for this endpoint.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource URI</td>
<td>Displays the endpoint relative resource URI entered on the Basic Info page.</td>
</tr>
<tr>
<td>Specify Query Parameters</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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 orchestrated 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.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select the attachment processing options</td>
<td>Configure the following options based on whether the request is inbound or outbound.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>• Accept attachments from request:</td>
<td>Select for the REST endpoint to process attachments from the inbound multipart request. This selection refreshes the page to display the Select the type of payload that you want the endpoint to receive field at the bottom of the page.</td>
</tr>
<tr>
<td>• Request is HTML form:</td>
<td>Select for the REST endpoint to accept to configure an HTML form. You must first select the Accept attachments from request option before you can select this option. This selection assumes that the media type is multipart/form-data.</td>
</tr>
<tr>
<td></td>
<td>For outbound (invoke) requests, select the multipart attachment type to include. This option is only available if you selected the POST action on the Basic Info page.</td>
</tr>
<tr>
<td>• Send attachments in request:</td>
<td>Select for the REST endpoint to process attachments from the outbound multipart request. This selection refreshes the page to display the Select the type of payload that you want the endpoint to receive field at the bottom of the page.</td>
</tr>
<tr>
<td>• Request is HTML form:</td>
<td>Select for the REST endpoint to accept to configure an HTML form. You must first select the Send attachments in request option before you can select this option. This selection assumes that the media type is multipart/form-data.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Select the request payload format</td>
<td>Note:</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• If you upload a schema file without a target namespace, a surrogate namespace is added to the schema file that all messages then use:</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>• XML Schema</td>
</tr>
<tr>
<td></td>
<td>• JSON Sample: Select this option to use Swagger and RAML files. JSON sample files of up to 100 KB in size are supported.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>• Sample XML Document (Single or No Namespace): Select this option to use an XML document to generate the schema.</td>
</tr>
<tr>
<td></td>
<td>• Sample JSON Document: Select this option to use a JSON document to generate the schema.</td>
</tr>
<tr>
<td>Schema Location</td>
<td>Specify the schema file in either of the following ways:</td>
</tr>
<tr>
<td></td>
<td>• Click <strong>Browse</strong> to select the request schema file to use.</td>
</tr>
<tr>
<td></td>
<td>• Click <strong>&lt;&lt;inline&gt;&gt;</strong> to copy and paste the JSON payload or URL into a text field. Click <strong>OK</strong> when complete.</td>
</tr>
<tr>
<td>Element</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Select the type of payload you want the endpoint to receive

- **None**: Select if no payload type is required.
- **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 an option in Select the 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.
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.
• Click the Delete icon to delete the row of a selected standard HTTP request header.

<table>
<thead>
<tr>
<th>HTTP Header Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Perform the following tasks:</td>
</tr>
<tr>
<td></td>
<td>• From the list, select the header to use.</td>
</tr>
</tbody>
</table>

Specify the custom HTTP request headers to use.

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.

<table>
<thead>
<tr>
<th>Custom Header Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enter the custom header name.</td>
</tr>
<tr>
<td>Custom Header Description</td>
<td>Enter an optional description.</td>
</tr>
</tbody>
</table>

REST Adapter Trigger CORS Configuration Page
Enter the REST Adapter CORS configuration properties for this endpoint.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowed Origins</td>
<td>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, <a href="http://localhost:8080">http://localhost:8080</a>, <a href="https://myhost.example.com:7002">https://myhost.example.com:7002</a>).</td>
</tr>
</tbody>
</table>

| 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.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select the attachment processing options</td>
<td>Configure the following options based on whether the request is inbound or outbound. For inbound (trigger) responses, select the multipart attachment type to include.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Accept attachments from response</strong>: Select to receive the response from the payload. This selection refreshes the page to display the <strong>Select the type of payload with which you want the endpoint to reply</strong> field at the bottom of the page.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Response is HTML form</strong>: Select for the REST endpoint to accept to configure an HTML form. You must first select the <strong>Accept attachments from response</strong> option before you can select this option. This selection assumes that the media type is multipart/form-data.</td>
</tr>
<tr>
<td></td>
<td>For outbound (invoke) responses, select the multipart attachment type to include.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Process attachments from response</strong>: Select for the REST endpoint to process attachments from the outbound multipart request. This selection refreshes the page to display the <strong>Select the type of payload with which you want the endpoint to reply</strong> field at the bottom of the page.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Response is HTML form</strong>: Select for the REST endpoint to accept to configure an HTML form. You must first select the <strong>Process attachments from response</strong> option before you can select this option. This selection assumes that the media type is multipart/form-data.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Select the response payload format</strong> Note:</td>
<td>• 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.</td>
</tr>
<tr>
<td>• If you upload a schema file without a target namespace, a surrogate namespace is added to the schema file that all messages then use:</td>
<td></td>
</tr>
<tr>
<td><a href="http://xmlns.oracle.com/cloud/adapter/nxsd/surrogate">http://xmlns.oracle.com/cloud/adapter/nxsd/surrogate</a></td>
<td>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.</td>
</tr>
<tr>
<td>• <strong>XML Schema</strong></td>
<td></td>
</tr>
<tr>
<td>• <strong>JSON Sample</strong>: 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.</td>
<td></td>
</tr>
<tr>
<td>• <strong>Sample XML Document (Single or No Namespace)</strong>: Select this option to use an XML document to generate the schema.</td>
<td></td>
</tr>
<tr>
<td>• <strong>Sample JSON Document</strong>: Select this option to use a JSON document to generate the schema.</td>
<td></td>
</tr>
<tr>
<td><strong>Schema Location</strong></td>
<td>Specify the schema file in either of the following ways:</td>
</tr>
<tr>
<td>• Click <strong>Browse</strong> to select the response schema file to use.</td>
<td></td>
</tr>
<tr>
<td>• Click <strong>&lt;inline&gt;</strong> to copy and paste the JSON payload or URL into a text field. Click <strong>OK</strong> when complete.</td>
<td></td>
</tr>
<tr>
<td><strong>Element</strong></td>
<td>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.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Select the type of payload with which you want the endpoint to reply | 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 an option in **Select the 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.
### 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.
- Click the **Delete** icon to delete the row of a selected standard HTTP response header.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Standard Response Headers</td>
<td>Select the standard HTTP response header to use from the default dropdown list.</td>
</tr>
<tr>
<td>HTTP Header Name</td>
<td>Perform the following tasks:</td>
</tr>
<tr>
<td></td>
<td>• From the list, select the header to use.</td>
</tr>
</tbody>
</table>

Specify the custom HTTP response headers to use.

### 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.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Custom Response Headers</td>
<td>Perform the following custom response header tasks:</td>
</tr>
<tr>
<td></td>
<td>• Click the <strong>Add</strong> icon to add custom HTTP response headers and optional descriptions.</td>
</tr>
<tr>
<td></td>
<td>• Click the <strong>Delete</strong> icon to delete the selected custom HTTP response headers.</td>
</tr>
<tr>
<td>Custom Header Name</td>
<td>Enter the custom header name.</td>
</tr>
<tr>
<td>Custom Header Description</td>
<td>Enter an optional description.</td>
</tr>
</tbody>
</table>

### REST Adapter Trigger Operation Selection Page

Enter the REST Adapter invoke operation selection parameters for this endpoint.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Object</td>
<td>Select the business object (resource) to use in this connection.</td>
</tr>
<tr>
<td>Operations</td>
<td>Select the operation (method) to perform on the business object in this connection.</td>
</tr>
</tbody>
</table>
Summary Page

You can review the specified adapter configuration values on the Summary page.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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 Back. Click Cancel to cancel your configuration details.</td>
</tr>
</tbody>
</table>

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 Header 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.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 does the endpoint perform? | 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.

Based on your selections, you can add parameters or configure a request and/or response for this endpoint | 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.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource URI</td>
<td>Displays the endpoint relative resource URI entered on the Basic Info page.</td>
</tr>
<tr>
<td>Specify Query Parameters</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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 orchestrated 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.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select the attachment processing options</td>
<td>Configure the following options based on whether the request is inbound or outbound.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>• Accept attachments from request: Select for the REST endpoint to process attachments from the inbound multipart request. This selection refreshes the page to display the Select the type of payload that you want the endpoint to receive field at the bottom of the page.</td>
</tr>
<tr>
<td></td>
<td>• Request is HTML form: Select for the REST endpoint to accept to configure an HTML form. You must first select the Accept attachments from request option before you can select this option. This selection assumes that the media type is multipart/form-data.</td>
</tr>
<tr>
<td></td>
<td>For outbound (invoke) requests, select the multipart attachment type to include. This option is only available if you selected the POST action on the Basic Info page.</td>
</tr>
<tr>
<td></td>
<td>• Send attachments in request: Select for the REST endpoint to process attachments from the outbound multipart request. This selection refreshes the page to display the Select the type of payload that you want the endpoint to receive field at the bottom of the page.</td>
</tr>
<tr>
<td></td>
<td>• Request is HTML form: Select for the REST endpoint to accept to configure an HTML form. You must first select the Send attachments in request option before you can select this option. This selection assumes that the media type is multipart/form-data.</td>
</tr>
</tbody>
</table>
**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:


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](#).
- **Sample XML Document (Single or No Namespace)**: Select this option to use an XML document to generate the schema.
- **Sample JSON Document**: Select this option to use a JSON document to generate the schema.
- **Raw**: 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:** Raw payload support is only available when the adapter is used as an invoke, not a trigger.

---

**Select the type of payload with which you want the endpoint to receive (if Raw payload format is selected)**

Select from a list of commonly used types provided in the dropdown menu. You can also select **Other Media Type** to provide a type not listed in the dropdown list—for example, `video/mp4`.

---

**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.
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.

Select the type of payload you want the endpoint to receive

- **None**: Select if no payload type is required.
- **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 an option in Select the 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 Invoke 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.
### 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.
- Click the **Delete** icon to delete the row of a selected standard HTTP request header.

<table>
<thead>
<tr>
<th>HTTP Header Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Perform the following tasks:</strong></td>
<td></td>
</tr>
<tr>
<td>• From the list, select the header to use.</td>
<td></td>
</tr>
</tbody>
</table>

Specify the custom HTTP request headers to use.

### 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.

<table>
<thead>
<tr>
<th>Custom Header Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enter the custom header name.</strong></td>
<td></td>
</tr>
<tr>
<td>Custom Header Description</td>
<td>Enter an optional description.</td>
</tr>
</tbody>
</table>

### REST Adapter Invoke Response Page

Enter the REST Adapter response payload details for the endpoint.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select the attachment processing options</td>
<td>Configure the following options based on whether the request is inbound or outbound. For inbound (trigger) responses, select the multipart attachment type to include.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Accept attachments from response</strong>: Select to receive the response from the payload. This selection refreshes the page to display the <strong>Select the type of payload with which you want the endpoint to reply</strong> field at the bottom of the page.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Response is HTML form</strong>: Select for the REST endpoint to accept to configure an HTML form. You must first select the <strong>Accept attachments from response</strong> option before you can select this option. This selection assumes that the media type is multipart/form-data.</td>
</tr>
<tr>
<td></td>
<td>For outbound (invoke) responses, select the multipart attachment type to include.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Process attachments from response</strong>: Select for the REST endpoint to process attachments from the outbound multipart request. This selection refreshes the page to display the <strong>Select the type of payload with which you want the endpoint to reply</strong> field at the bottom of the page.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Response is HTML form</strong>: Select for the REST endpoint to accept to configure an HTML form. You must first select the <strong>Process attachments from response</strong> option before you can select this option. This selection assumes that the media type is multipart/form-data.</td>
</tr>
</tbody>
</table>
**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:  
  
  ```xml  
  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.  
- **Sample XML Document (Single or No Namespace)**: Select this option to use an XML document to generate the schema.  
- **Sample JSON Document**: Select this option to use a JSON document to generate the schema.  
- **Raw**: 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`.  
  
Select the type of payload with which you want the endpoint to reply (if **Raw** payload format is selected) | Select from a list of commonly used types provided in the dropdown menu. You can also select **Other Media Type** to provide a type not listed in the dropdown list—for example, `video/mp4`.  
  
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.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Select the element that defines the payload structure.</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
| **Select the type of payload with which you want the endpoint to reply.** | 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 an option in **Select the 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 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.
### 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.
- 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.

### 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.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Business Object</strong></td>
<td>Select the business object (resource) to use in this connection.</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td>Select the operation (method) to perform on the business object in this connection.</td>
</tr>
</tbody>
</table>
Summary Page

You can review the specified adapter configuration values on the Summary page.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 Back. Click Cancel to cancel your configuration details. |
Implement Common Patterns Using the REST Adapter

You can use the REST Adapter to implement the following common patterns.

Topics:

- Configure the REST Adapter to Consume a REST API Protected with 2-Legged OAuth Token-Based Authentication
- Configure the REST Adapter to Consume a REST API Protected with 3-Legged OAuth Token-Based Authentication
- Understand Security Configurations for Invoking Popular OAuth-Protected APIs
- Configure the REST Adapter to Consume a REST API Protected with the API Key
- Configure the REST Adapter to Consume a REST API Protected with OAuth 1.0a One-Legged Authentication
- Allow Client Applications to Consume an Integration Exposed as an OAuth-Protected REST API
- Override the Endpoint URI/Host Name for an External REST API at Runtime
- Map to Construct the Payload for an External REST API that Accepts multipart/form-data as the Content Type
- Configure the REST Adapter to Consume an External REST API Described Using a Swagger Document
- Configure the REST Adapter to Consume an External REST API Described Using a RAML Document
- Configure the REST Adapter to Consume an External REST API with No Metadata Described in a Document
- 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
- Map JSON when the REST Adapter Request is Configured with multipart/form-data
- 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
- Configure the REST Adapter to Expose an Integration as a REST API
- 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
- Enter q as a Standard HTTP Query Parameter with the Query as a Value
- JSON to XML Special Character Conversion
Configure the REST Adapter to Consume a REST API Protected with 2-Legged OAuth 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 in the Credentials dialog of the Connections page for the REST Adapter.

![Security Policy Options](image)

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 two-legged OAuth flows called as an **OAuth Custom Two Legged Flow**.

OAuth 2.0 specification defines the following four OAuth flows:

- **OAuth Client Credentials**
- **OAuth Resource Owner Password Credentials**
- **OAuth Implicit Grant Authorization**
- **OAuth Authorization Code Credentials**

The **OAuth Client Credentials** and **OAuth Resource Owner Password Credentials** options are categorized as two-legged OAuth 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
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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-X</td>
<td>GET</td>
<td>PUT</td>
<td>POST</td>
</tr>
<tr>
<td>-H</td>
<td>-H &quot;key: value&quot;</td>
<td>Add each header key value pair as described. There can be multiple headers.</td>
<td>No</td>
</tr>
<tr>
<td>-d</td>
<td>-d 'data-as-string'</td>
<td>String data enclosed within single quotes. Escape any quotes within the data string.</td>
<td>No</td>
</tr>
<tr>
<td>URI</td>
<td>Uri (within quotes)</td>
<td>- -</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**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:

```
```

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.

By default, the $variables are mapped to property names containing relevant tokens as follows:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Mapping to a Property with Name</th>
<th>Example Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$access_token</td>
<td>access.[tT]oken</td>
<td>access_token</td>
</tr>
<tr>
<td>$refresh_token</td>
<td>refresh.[tT]oken</td>
<td>refresh_token</td>
</tr>
<tr>
<td>$token_type</td>
<td>token.?{tT}ype</td>
<td>token_type</td>
</tr>
<tr>
<td>$expiry</td>
<td>expires_in</td>
<td>expires_in</td>
</tr>
</tbody>
</table>

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:

```json
{
    "access_token": "1-253912-240049694-f85c1d679211c",
    "expiry": 21599,
    "token_type": "Bearer",
    "extended_token": "5707efdf04912f53b61c5ec5dc7f166"
}
```
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.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Default Mapping to a Property with Name</th>
<th>Example Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$refresh_token</td>
<td>extended_token</td>
<td>extended_token</td>
</tr>
<tr>
<td>$expiry</td>
<td>Expiry</td>
<td>Expiry</td>
</tr>
</tbody>
</table>

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.

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}
```

<table>
<thead>
<tr>
<th>Access Token Usage</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-H Authorization: $ {token_type} $ {access_token}</td>
<td>The access token is passed as a header at runtime for accessing the protected resource.</td>
<td>-H Authorization: Bearer AASDFADADX</td>
</tr>
<tr>
<td>?api_key=${access_token}</td>
<td>The access token is passed as a query parameter at runtime for accessing the protected resource.</td>
<td><a href="http://someapi.com/employee?api_key=ASDFADADX">http://someapi.com/employee?api_key=ASDFADADX</a></td>
</tr>
</tbody>
</table>

**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:

```
```

This request contains a variable that is replaced with the actual value of the current refresh token at runtime.

**Configure the REST Adapter to Consume a REST API Protected with 3-Legged OAuth 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.
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The user specifies the authorization request URI. The user is redirected by the user agent (browser) to the authorization URI.</td>
</tr>
<tr>
<td>2</td>
<td>The resource owner logs in to authenticate and provide consent to the client application to access its resources.</td>
</tr>
<tr>
<td>3</td>
<td>The authorization server sends a callback request to the client application and sends the authorization code.</td>
</tr>
<tr>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>The authorization server responds to the access token request by sending an access token to the client application.</td>
</tr>
<tr>
<td>6</td>
<td>The client application uses the access token to make requests for protected resources.</td>
</tr>
</tbody>
</table>

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

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 \( \text{${refresh\_token}} \) 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 \$\text{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 \$\text{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 \texttt{cURL}, but it is more basic and only supports the following options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-X</td>
<td>GET</td>
<td>The HTTP verb in the access token request.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-H</td>
<td>-H &quot;key: value&quot;</td>
<td>Add each header key value pair as described. There can be multiple headers.</td>
<td>No</td>
</tr>
<tr>
<td>-d</td>
<td>-d 'data-as-string'</td>
<td>String data enclosed within single quotes. Escape any quotes within the data string.</td>
<td>No</td>
</tr>
<tr>
<td>URI</td>
<td>Uri (within quotes)</td>
<td>- -</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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:

```bash
-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_SECRET&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:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Mapping to a Property with Name</th>
<th>Example Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$auth_code</td>
<td>code</td>
<td>code</td>
</tr>
<tr>
<td>$access_token</td>
<td>access.[tT]oken</td>
<td>access_token</td>
</tr>
<tr>
<td>$refresh_token</td>
<td>refresh.[tT]oken</td>
<td>refresh_token</td>
</tr>
<tr>
<td>$token_type</td>
<td>token.[tT]ype</td>
<td>token_type</td>
</tr>
<tr>
<td>$expiry</td>
<td>expires_in</td>
<td>expires_in</td>
</tr>
</tbody>
</table>

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.

Understand Security Configurations for Invoking Popular OAuth-Protected APIs

The following blog provides the configuration details for consuming some popular OAuth-protected APIs.
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.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The resource owner authenticates and generates an API key for the given client application.</td>
</tr>
<tr>
<td>2</td>
<td>The resource owner shares the generated API key with the client application.</td>
</tr>
<tr>
<td>3</td>
<td>The client application makes a request for a resource using the API key.</td>
</tr>
</tbody>
</table>

When you select Configure Security on the Connections page for a REST Adapter, you select **API Key Based Authentication**.
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 Invoke Connections](#) for details.

At runtime, the API key is automatically passed to the endpoint while sending the request.

## Configure the REST Adapter to Consume a REST API Protected with OAuth 1.0a One-Legged Authentication

This section provides an overview of the OAuth 1.0a 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 1.0a 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).
- **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 the 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.
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. Acquiring the token requires a 3-legged OAuth flow.

The REST API describes the client application registration and flow. See the REST API for Oracle Integration.

Use the following steps to acquire a token and invoke an Oracle Integration flow.

**Step 1:** Register an application with Oracle Identity Cloud Service.

1. Log in to Oracle Identity Cloud Service. You need the Oracle Identity Cloud Service Admin role or the Application Admin role to perform these steps.
2. Click the icon with a rectangle and a plus sign on the upper right of the Applications panel.
3. In the Add Application dialog, click **Trusted Application**.
4. Enter the name of the application in the **App Details** field.
5. In the Add Trusted Application dialog:
   - Select the **Configure this application as a client now** radio button.
   - Check the **Refresh Token** and **Authorization Code** boxes.
   - Provide a valid URL in the **Redirect URL** field.
6. In the **Accessing APIs From Other Applications** section of the same page, click the **Add** button to invoke the Add Scope dialog.
7. In the Add Scope dialog, select an available resource and click the right arrow button.
8. Select one or more scopes, then click **Add**.
9. Click **Next** and **Finish**.
10. Make a note of the **Client ID** and **Client Secret**.

**Step 2:** OAuth Flow - Get code

1. Access the following URL in a browser using the client ID from the previous step. Once you log in you are redirected to a redirect URL that is invalid in this case.
You can see code from the URL in the browser. Note that the postman application gets the access token in the following example.

https://IDCS_URL/oauth2/v1/authorize?clientId=client_Id&response_type=code&scope=scope&redirect_uri=redirect_Uri

The scope is related to a service instance like:

https://A22222222222222222222222222222222.instance.com:443urn:opc:resource:consumer::all

For example:


Note the following:

- You provide this URL when configuring the trusted application in Oracle Identity Cloud Service.
- The default value for access token expiration is 3600 seconds. This value can be configured in the trusted application in Oracle Identity Cloud Service.
- The redirect URL is the URL of the client application invoked by Oracle Identity Cloud Service to send authorization code to the client.

Step 3: OAuth Flow - Get access token

1. Use the code obtained in Step 2 in the following curl command to get the access token. The access token is available in the response and is valid for 60 seconds.

   curl -u 'clientId:clientSecret' https://idcs_url/oauth2/v1/token -d 'grant_type=authorization_code&code=code'

For example:

   curl -u '111111111111111111111111111111111111111111:4c189d50-f78a-49b5-86c6-04b734cbaddc' https://idcs-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.identity.alprd1.oc9prd.com/oauth2/v1/token -d 'grant_type=authorization_code&code=AQIDBAUEfZVyq8PsWU6GmM-THFlptWUGZF-RkcOMG1R2C0tjVmpBe2LYrABNAhhyb44zVgPcvqsa8Adj28VZMKNoMTEgRU5DUl1QVE1PT19LRVxxNCB7djF9NCA='
Step 4: Invoke Oracle Integration Flow

1. Use the access token to invoke any Oracle Integration endpoint as follows:

```
```

For example:

```
curl -v -H "Accept:application/json" -H "Authorization: Bearer eyJ4NXXqi1N1i6ImpI
aEhFWDdFTW5sMWtQRktickdycEpmWXi1MFMeURWa0IUREZkdktDq0iLCJ4NXQioIj3Yz20bH
NVEtWLXZ
jMkm2T2dVR1dJU2hyUnMILcJrAqjOiJTSUdOSU5HX0tFWSIsImFsZyI6Il16LTMjU2IIn0.eyJ1c2
VyX3R6IjoiQW11cm1jYyYWvQ2hpY2FnbyIsIn11Yi16InBoAwxpcC52YXJnaGVzZUBvcmFjbGUy29tI
widXNlc19s
2NhGUkO1Jbi1SIsInZXfjfZGxZgxeW5hbWU1OiJQaGlSaXgMy2z1c2UIClJC12VyInRk
bFudCu5u
YW11IjoiAWrjcy1mOGQ2Yzg1YxYMTU0ZTIT2OTFhMGU0ZTFkzjQyODg1ZiIsImZci6ImZci6ImZhi
li6icV3
ixZ1ihcHbpbmdhdHh1IjoidXN1ck5hbWUIClJc3MiOiJodHRzwpcLlwvARlbawRlbwRpdHku
b3JhY2xvYXQ2
QuY29tXC8iLCJc2btdfHl2WS11KWFU1idXNlc190Z5hbWU1IjoiAWrjcy1mOGQ2Yzg1Y
zyXMTU0Z
TI20TFRhMGU0ZTFKzjQyODg1ZiIsImNsaWVudFd9pZCI6IlJTMjU2In0.eyJ1cl90
ZW5hbnRuYW11IjoiAWrjcy1mOGQ2Yzg1Y
zXMTU0Z
NDJhIi6ic2liKjo1NWYwMjgzZGYtYzF1OSOS2TFlLTkxYjgtYjcwMDIyMjhiMDIyIiwic3V
iX21hcHBpbnRhdHh1IjoidXN1ck5hbWUIClJc3MiOiJodHRzwpcLlwvARlbawRlbwRpdHku
b3JhY2xvYXQ2
```

Chapter 5

Allow Client Applications to Consume an Integration Exposed as an OAuth-Protected REST API
Sample Application

<<%--This file is subject to the terms and conditions defined in file 'LICENSE.MD' which is part of this source code package.--%>
% page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
import="com.example.utils.ClientConfig, com.example.beans.UserBean, org.codehaus.jettison.json.JSONObject"

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<title>Address Finder</title>
<meta name="description" content="A sample code">
<meta name="viewport" content="width=device-width, initial-scale=1">
<script src="js/jquery.min.js"></script>
<script src="js/scripts.js"></script>
<link rel="stylesheet" href="css/style.css">
<link rel="stylesheet" href="css/font-awesome.min.css">
</head>

<script>
$(document).ready(function() {
  $('#myConsole').click(function(){
    var url = '<%= ClientConfig.MYCONSOLE_UI_URL %>';  
    window.open(url);
  });
  $('#adminConsole').click(function(){
    var url = '<%=ClientConfig.ADMIN_UI_URL%>';  
    window.open(url);
  });
  $('#loginButton').click(function() {
    window.location.href='getResource?resource=openid';
  });
  $('#logoutlink').click(function() {
    var url = '<%=ClientConfig.LOGOUT_SERVICE_URL%>'
    post_logout_redirect_uri=''
    =ClientConfig.APP_POST_LOGOUT_REDIRECT_URI
    +"&id_token_hint="+session.getAttribute("idToken")';
    window.location.href = url;
  });
window.addEventListener("DOMContentLoaded", function () {var form = document.getElementById("search");

  document.getElementById("submit").addEventListener("click", function () {

Chapter 5
Allow Client Applications to Consume an Integration Exposed as an OAuth-Protected REST API

ijkl2FCygmnmBzfOSFVe1L0xSwk44r_ujytJlgBRnXEHRRLXzgpi1YjRTMF1bI00g2MR8rCFu3IO
d8ib4p
Ke8VRkhTdeNt3IJS-1k88yYG65qRv-dVD2pfwR4vlCqpq-
PihXRq1AfdyItcTukZqVYTF6PlvMEIj6JH55
UMkwzC503u2F2daGhgmDZqV6y-vUTxtXQxgXr5Kw" https://myhost.com/ic/api/
integration/v1/flows/rest/ECHOINTEGRATION/1.0/echo/HelloWorld

<!--This file is subject to the terms and conditions defined in file 'LICENSE.MD' which is part of this source code package.--%>
% page language="java" contentType="text/html; charset=ISO-8859-1"
pag
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 Request is HTML Form on the Request page of the Adapter Endpoint Configuration Wizard or when you select Response is HTML Form 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 provides 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:

\[
\text{Content-Transfer-Encoding} := \text{"BASE64"} / \text{"QUOTED-PRINTABLE"} / \\
\text{"8BIT"} / \text{"7BIT"} / \\
\text{"BINARY"} / \text{x-token}
\]

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 1 - source and target both have multipart requests with JSON/XML payload (Category A)**

The following sample map focuses only on the mapping of **attachmentReference** to the target. 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 do that.

**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. For example, in the following image, the target endpoint is expecting a PDF document under a body part named `file` in the multipart/form-data request payload.

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.

Scenario 4 - source is an FTP file read operation (nonmultipart) and outbound is multipart/mixed with a JSON or XML payload
Configure the REST Adapter to Consume an External REST API Described Using a Swagger Document

Oracle Integration can seamlessly integrate with REST APIs described using Swagger. The following example shows how to consume a Swagger-based REST API.

1. Create a REST Adapter connection by selecting Swagger Definition URL in the Connection Type field and specifying the URL pointing to the Swagger definition in the Connection URL field.
2. If the Swagger resource is protected, provide the necessary security configuration and test and save the connection.
3. Design an integration using the REST Adapter as an invoke connection. When the connection is used as an invoke, the REST Adapter automatically lists the operations and resources from the Swagger definition.
4. Select the business object and the operation to invoke the object on the Operation Selection page of the Adapter Endpoint Configuration Wizard.
5. Complete the wizard and the mappings.
6. Activate and invoke the flow.

Configure the REST Adapter to Consume an External REST API Described Using a RAML Document

Oracle Integration can seamlessly integrate with REST APIs described using RAML. The following example shows how to consume a RAML-based REST API.

1. Create a REST Adapter connection by selecting RAML Definition URL in the Connection Type field and specify the URL pointing to the RAML definition in Connection URL field.
2. If the resource is protected, provide the necessary security configuration and test and save the connection.
3. Design an integration using the REST Adapter as an invoke connection. When the connection is used as an invoke, the REST Adapter automatically lists the operations and resources from the definition.

4. Select the business object and the operation to invoke the object on the Operation Selection page of the Adapter Endpoint Configuration Wizard.

5. Complete the wizard and the mappings.

6. Activate and invoke the flow.

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.

The API is described at [https://carbon-intensity.github.io/api-definitions/#intensity](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:

```bash
curl -X GET https://api.carbonintensity.org.uk/intensity/date -H 'Accept: application/json'
```

The response is of the form:

```json
{"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.

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](https://api.carbonintensity.org.uk).

The following steps describe how to configure the relative REST 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.

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.

A JSON sample can also be provided using the <<<inline>>> option.
5. Complete the rest of the Adapter Endpoint Configuration Wizard.
6. Complete the mappings.

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.
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 Send attachments in request and Request is HTML form check boxes are selected on the Request page).

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.
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`.

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 orchestrated integration, the FTP adapter is only supported as an invoke connection. Implement this use case by selecting either an App Driven Orchestration
or **Scheduled Orchestration** integration on the Create Integration - Select a Style dialog.

The following example provides a high level overview on how to implement this use case as a scheduled orchestration.

1. Configure an FTP Adapter with the **List Files** operation to list files from an SFTP server.
2. Configure a for-each action to iterate through the **List Files** operation response.

3. Configure a second FTP Adapter with the **Read Files** operation to read individual files inside the loop.

4. Configure the FTP Adapter to not specify a structure of the file.
5. Configure a REST Adapter. The reference of the **Read Files** operation is handed over to the outbound REST Adapter.

6. Configure the REST Adapter payload as **application/octet-stream**.
7. Configure the mapper to read individual files in the for-each loop.

8. Configure the mapper to send the read file to the outbound REST Adapter.

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.

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 the **Configure Custom Request Header** check box.
   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, click **Configure Connectivity**. 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. Click **Configure Security**.
5. From the **Security Policy** list, select **AWS Signature Version 4**.
6. Specify the following details.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Key</td>
<td>Enter the access key obtained when you created your Amazon security credentials.</td>
</tr>
</tbody>
</table>
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.

**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:

```json
{
   "_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:
<table>
<thead>
<tr>
<th>Special Character</th>
<th>Converted Value Represented in the Mapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; &quot;</td>
<td><em>0x737063</em></td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td><em>0x736c68</em></td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td><em>0x626c68</em></td>
</tr>
<tr>
<td>&quot;:&quot;</td>
<td><em>0x636c6e</em></td>
</tr>
<tr>
<td>&quot;;&quot;</td>
<td><em>0x73636e</em></td>
</tr>
<tr>
<td>&quot;(&quot;</td>
<td><em>0x6c7072</em></td>
</tr>
<tr>
<td>&quot;)&quot;</td>
<td><em>0x727072</em></td>
</tr>
<tr>
<td>&quot;&amp;&quot;</td>
<td><em>0x616d70</em></td>
</tr>
<tr>
<td>&quot;,&quot;</td>
<td><em>0x636d61</em></td>
</tr>
<tr>
<td>&quot;#&quot;</td>
<td><em>0x706e64</em></td>
</tr>
<tr>
<td>&quot;?&quot;</td>
<td><em>0x717374</em></td>
</tr>
<tr>
<td>&quot;&lt;&quot;</td>
<td><em>0x6c7374</em></td>
</tr>
<tr>
<td>&quot;&gt;&quot;</td>
<td><em>0x677274</em></td>
</tr>
<tr>
<td>&quot;start&quot;</td>
<td><em>0x737472</em></td>
</tr>
<tr>
<td>Special Character</td>
<td>Converted Value Represented in the Mapper</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td><code>@</code></td>
<td><em>0x617472</em></td>
</tr>
<tr>
<td><code>$</code></td>
<td><em>0x646c72</em></td>
</tr>
<tr>
<td><code>{</code></td>
<td><em>0x6c6362</em></td>
</tr>
<tr>
<td><code>}</code></td>
<td><em>0x726362</em></td>
</tr>
<tr>
<td><code>%</code></td>
<td><em>0x706572</em></td>
</tr>
</tbody>
</table>
Troubleshoot the REST Adapter

Review the following topics to learn about troubleshooting issues with the REST Adapter.

Topics:

• REST Services that Return Multiple Successful Responses
• Error Handling with the REST Adapter
• REST Service Invoked by the REST Adapter Returns a 401 Unauthorized Status Response
• Configuration Limitation of Ten Pages in the Adapter Endpoint Configuration Wizard
• Keys with Null Values During JSON Transformation are Removed
• Large Sample JSON File Processing with Special Characters
• SSL Certification Troubleshooting Issues
• Fault and Response Pipeline Definitions in Basic Routing Integrations
• Empty Arrays Are Not Supported in Sample JSON Files
• Invoke Endpoint URI Must Match the Base URI + Resource URI in REST Adapter
• JD Edwards Form Service Invocation with the REST Adapter Causes APIInvocation Error
• REST Adapter Data is Only Saved When You Click Next
• Convert XML to a JSON Document
• Supported Special Characters in JSON Samples
• content-type is Missing for an Asynchronous Flow
• REST URLs Exceeding 8251 Characters Fail

Additional integration troubleshooting information is provided. See Troubleshoot Oracle Integration in *Using Integrations in Oracle Integration*.

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.

<table>
<thead>
<tr>
<th>*APIInvocationError</th>
</tr>
</thead>
<tbody>
<tr>
<td>*type</td>
</tr>
<tr>
<td>*title</td>
</tr>
<tr>
<td>*detail</td>
</tr>
<tr>
<td>*errorCode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>*errorDetails</th>
</tr>
</thead>
<tbody>
<tr>
<td>*type</td>
</tr>
<tr>
<td>*instance</td>
</tr>
<tr>
<td>*title</td>
</tr>
<tr>
<td>*errorPath</td>
</tr>
<tr>
<td>*errorCode</td>
</tr>
</tbody>
</table>

The errorDetails section contains the actual cause.

You can handle the APIInvocationError with a fault handler in the orchestrated 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.

<table>
<thead>
<tr>
<th>Condition</th>
<th>HTTP Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid client request</td>
<td>4xx</td>
<td>There are several conditions that can cause client side failures, including:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An invalid resource URL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Incorrect query parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An unsupported method type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An unsupported media type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bad data</td>
</tr>
<tr>
<td>Downstream processing errors</td>
<td>5xx</td>
<td>All other errors that can occur within the integration, including:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An invalid target</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An HTTP error response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• General processing errors</td>
</tr>
</tbody>
</table>

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:

```json
{
    "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                        }",
    "title" : "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:

![Map: fault to APIInvocationError](image)

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.

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Workaround</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>If the wizard does not refresh while configuring multiple operations, click <strong>Back</strong> to return to a previous page and then press <strong>Next</strong> to refresh to the current page.</td>
<td>The REST Adapter multiple sources per endpoint use case requires multiple iterations over the same sets of pages. This is currently a technical restriction.</td>
</tr>
</tbody>
</table>

### 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:

```json
{
    "input": "input",
    "val": null,
    "response": "response"
}
```

Oracle Integration sends the outbound request with the following JSON output.

```json
{
    "input": "input",
    "response": "response"
}
```

If you need the key available at the outbound service, use the following payload:

```json
{
    "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.

<table>
<thead>
<tr>
<th>Condition</th>
<th>HTTP Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid client request</td>
<td>4xx</td>
<td>There are several conditions that can cause client side failures, including:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Invalid resource URL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Incorrect query parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Unsupported method type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Unsupported media type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bad data</td>
</tr>
<tr>
<td>Downstream processing errors</td>
<td>5xx</td>
<td>All other errors that can occur within the integration, including:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Invalid target</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• HTTP error response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• General processing errors</td>
</tr>
</tbody>
</table>

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:

```json
{
   "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" : "{"error_message": "Invalid request. Missing the 'origin' parameter.",
      "routes" : [],
      "status" : "INVALID_REQUEST",
      "title" : "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:

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.
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.us.oracle.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.el.jdemf.FormRequest["formInputs"]", "exception": "com.fasterxml.jackson.databind.JsonMappingException",}}]]
```
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:

```xml
<element name="phone" type="string"/>
```

At runtime:

• XML

```xml
<Phone>23249480</Phone>
```

• JSON

```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.
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

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 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.

When complete, the integration looks as follows.

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.