Informatica®

Informatica® Cloud Application Integration
April 2024

Invoke

Informatica Cloud Application Integration Invoke
April 2024

Publication Date: 2024-04-04

Table of Contents

Preface

Use *Invoke* to learn how to deploy Application Integration assets.

CHAPTER 1

Runtime tasks for processes

After you create a process, you can perform the following runtime tasks:

Publish the process

You must publish a process to invoke or run the process. When you publish a process, Application Integration generates service endpoint URLs, a Swagger file, an OpenAPI 3.0 file, and a WSDL file for the process. You can use the service endpoint URLs to invoke the process. You can use the Swagger, OpenAPI 3.0, and WSDL files to view the API definitions of the process.

Invoke the process

You can invoke the process by deploying it to the Cloud Server, a specific Secure Agent, or a Secure Agent group. You can use different HTTP verbs to invoke a process.

Create process inputs and run the process with process inputs

After you publish a process and create process inputs, you can run the process with process inputs to test it. After you run the process, you can view details of the successful and unsuccessful executions of the process instances.

Publishing a process

You must publish a process to invoke or run the process.

You can also publish multiple processes in bulk. For more information, see Chapter 3, “Publishing Application Integration assets in bulk” on page 28.

When you publish a process, Application Integration activates the API. It generates service URLs, a Swagger file, an OpenAPI 3.0 file, and a WSDL file for the process. You can use the service endpoint URLs to invoke the process. You can use the Swagger, OpenAPI 3.0, and WSDL files to view the API definitions of the process.

If you edit a published process, you must publish the process again for the changes to get reflected. Otherwise, the process status changes to **Outdated** in the **Property Details** dialog box and on the **Explore** page. The **Outdated** status indicates that the process contains unpublished changes.

1. On the **Explore** page, navigate to the process that you want to publish and click **Publish**.

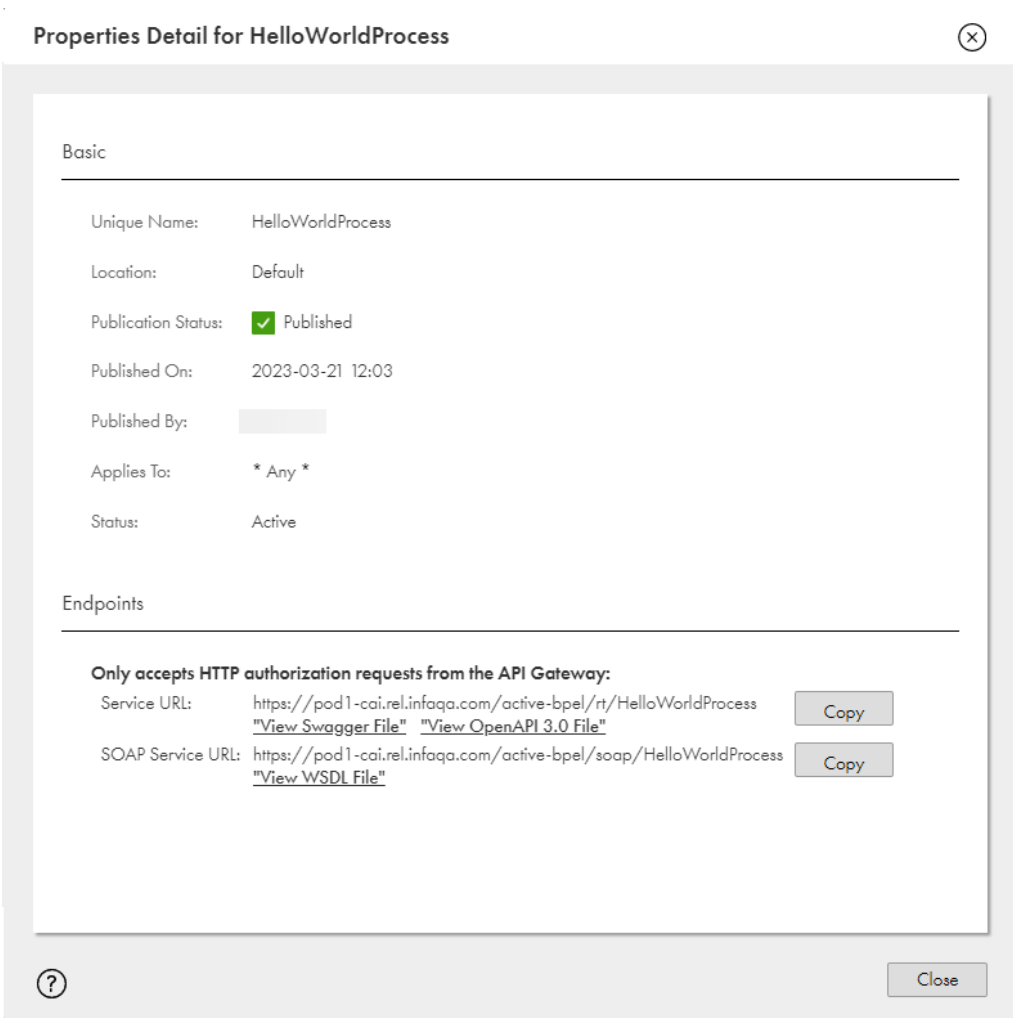
   Application Integration publishes the process and activates the API. It generates the service URLs, a Swagger file, an OpenAPI 3.0 file, and a WSDL file.

   Note: If you do not specify the allowed groups or allowed users, or allow anonymous access for the process, Application Integration does not generate the service URLs, Swagger file, OpenAPI 3.0 file, and WSDL file.

2. On the **Actions** menu, click **Properties Detail** to view the API status, generated service URLs, Swagger file, OpenAPI 3.0 file, and WSDL file.

   The **Properties Detail** dialog box appears.

   The following image shows the **Properties Detail** dialog box:



Properties Detail for HelloWorldProcess

Basic

Unique Name: HelloWorldProcess

Location: Default

Publication Status: ✓ Published

Published On: 2023-03-21 12:03

Published By:

Applies To: * Any *

Status: Active

Endpoints

Only accepts HTTP authorization requests from the API Gateway:

Service URL: https://pod1-cai.rel.infaqa.com/active-bpel/rt/HelloWorldProcess
"View Swagger File" "View OpenAPI 3.0 File"

SOAP Service URL: https://pod1-cai.rel.infaqa.com/active-bpel/soap/HelloWorldProcess
"View WSDL File"

The status indicates that the API is active.

Note: When you publish a process from Application Integration for the first time, the API is activated and displayed on the **APIs** page in Application Integration Console. Later, when you activate or deactivate the API from the **APIs** page, Application Integration also updates the API status in the **Properties Detail** dialog box. Republishing a process with an inactive API status does not activate the API. You must activate the API from the **APIs** page to make the API available for the user. For more information about activating and deactivating the API, see *APIs* in the Monitor help.

You can also view the service URL and SOAP service URL, and use them to invoke the process. The **Endpoints** section indicates if the process is configured to only accept HTTP authorization requests from the API Gateway.

You can click the **View Swagger File**, **View OpenAPI 3.0 File**, and **View WSDL File** links to view the associated Swagger file, OpenAPI 3.0 file, and WSDL file.

Unpublishing a process

To edit the process name or API name, or disable a published process, you must unpublish the process. You cannot call an unpublished process from the service endpoint URLs. Therefore, after you unpublish a process, you must update the associated API clients accordingly.

1. On the **Explore** page, navigate to the process that you want to unpublish.
2. On the **Actions** menu, click **Unpublish**.

   Application Integration unpublishes the process and removes the API from the **APIs** page in Application Integration Console. It disables the service URLs, Swagger file, OpenAPI 3.0 file, and WSDL file.

After you make the necessary changes, publish the process again to activate the API, generate the service URLs, Swagger file, OpenAPI 3.0 file, and WSDL file. After you publish the process, the process is enabled again.

Invoking Processes Deployed to the Cloud Server

The following topics describe how you can invoke Process Designer and Process Developer processes deployed to the Cloud Server.

Web Services and SOAP Endpoints

Process Designer exposes web services through an automatically generated WSDL interface that you can access and use as a SOAP endpoint. Each web service you interact with contains a set of methods, also called operations, hosted on an application server. You can invoke these operations remotely in the cloud or over a network. When you invoke an operation, you create a SOAP message and send it to the web service, typically over HTTP/ HTTPS. SOAP is a commonly used XML-based messaging protocol used to exchange information.

A web service has a WSDL (Web Services Definition Language) document, an XML description of the operations and how to invoke them. Every web service you want to interact with must have a service name and an endpoint URL (the location to which you send SOAP messages in order to invoke the operation).

The caller of the web service need not know anything about the internal details, as the SOAP interface publishes the available operations. For each call, the web service returns a response message. The response contains either the information requested or fault information, in case of an error.

This simple SOAP message calls the "Add" method and returns the answer:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:s='http://www.w3.org/2001/XMLSchema'>
    <SOAP-ENV:Body>
        <Add xmlns="http://www. math.org">
            <arg1 xsi:type="s:decimal">100</arg1>
            <arg2 xsi:type="s:decimal">25</arg2>
        </Add>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In this case, the response message is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
```

```
    xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:s='http://www.w3.org/2001/XMLSchema'>
    <SOAP-ENV:Body>
        <AddResponse xmlns="http://www.math.org">
            <AddResult>125</AddResult>
        </AddResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Definition of Processes Exposed as Web Services

When you define processes with Process Designer, the SOAP endpoint is available for use by SOAP clients that handle the "xsd:any" payload style, which means the clients can process inputs, outputs and handle faults.

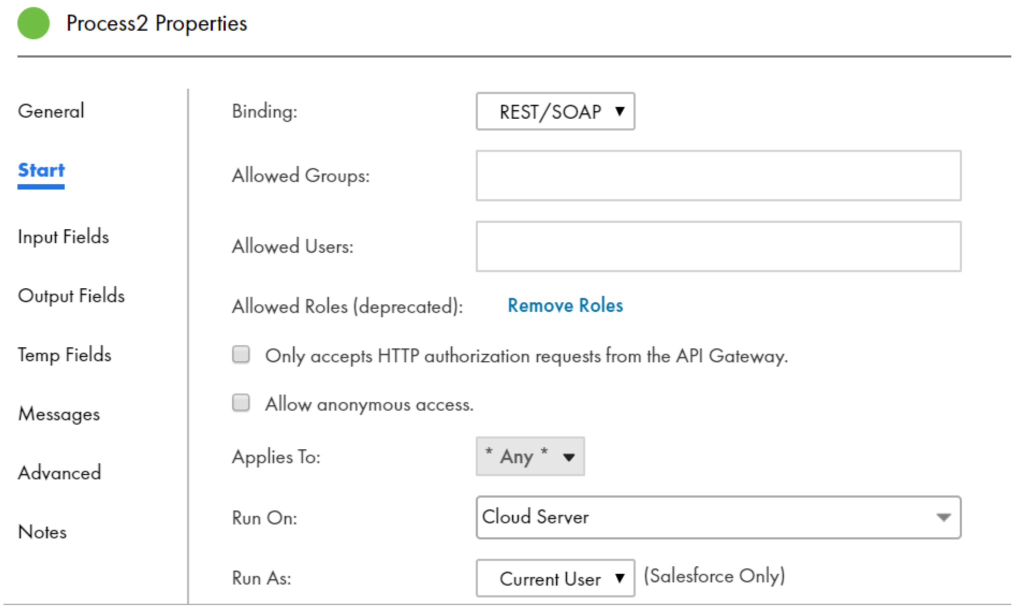
Process Designer supports the SOAP 1.1 standard with the document/literal message style and encoding option used by most web services.

With the WSDL interface, you can:

- Define WSDL input and output types with an inline schema based on the process input and output fields or process objects. The namespace is automatically generated.
- Display the SOAP Service URL in the process details.
- Link to the WSDL definition from the process details.
- Use basic authentication, consistent with REST.
- Generate faults using a Throw step, which allows you to share a single fault and provide a description.
- Generate correlation properties in the WSDL to handle inline message events and boundary conditions.

To define a SOAP endpoint in Process Designer:

1. In the process properties, choose **REST/SOAP** as the Start **Binding**.
   The following image shows the **Binding** property set as REST/SOAP:



Binding: REST/SOAP

Allowed Groups:

Allowed Users:

Allowed Roles (deprecated):

□ Only accepts HTTP authorization requests from the API Gateway.

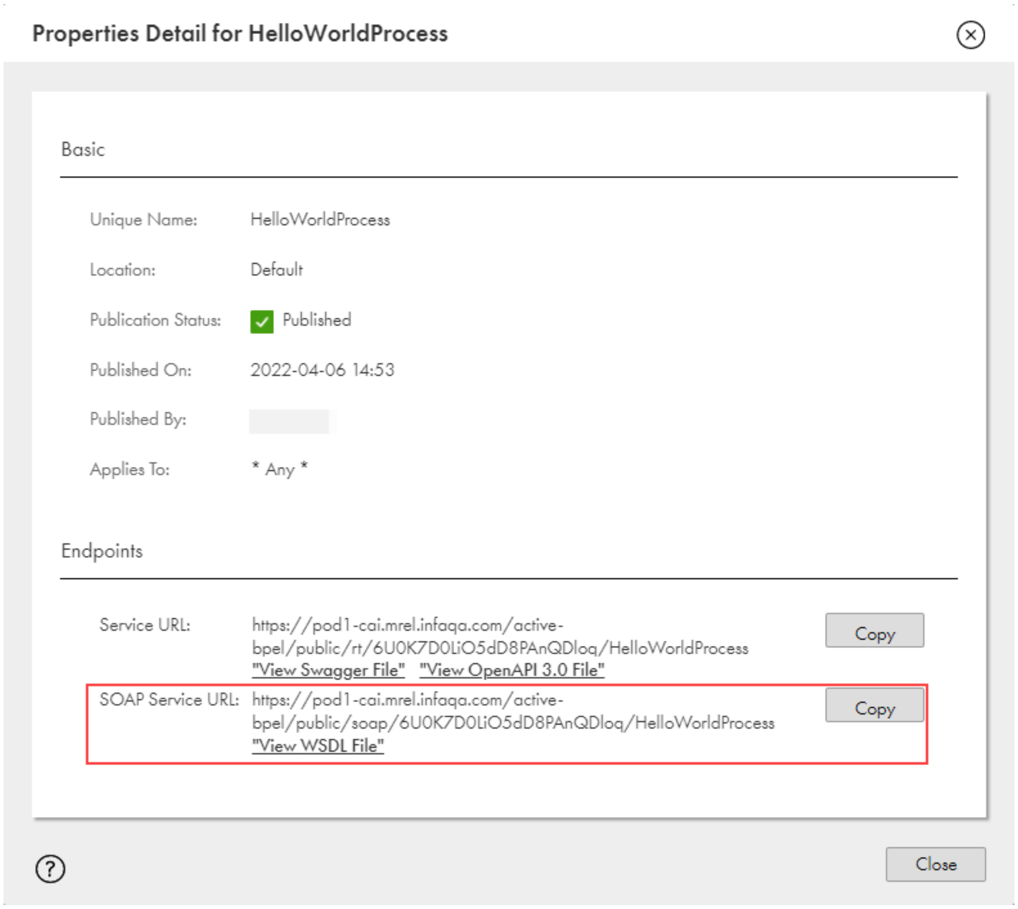
□ Allow anonymous access.

Applies To: * Any *

Run On: Cloud Server

Run As: Current User (Salesforce Only)

2. After you define and publish the process, you can see the SOAP Service URL.
   The following image shows the SOAP Service URL:



Properties Detail for HelloWorldProcess

Basic

Unique Name: HelloWorldProcess

Location: Default

Publication Status: ✓ Published

Published On: 2022-04-06 14:53

Published By: [redacted]

Applies To: * Any *

Endpoints

Service URL: https://pod1-cai.mrel.infaqa.com/active-bpel/public/rt/6U0K7D0LiO5dD8PAnQDloq/HelloWorldProcess
"View Swagger File" "View OpenAPI 3.0 File"

SOAP Service URL: https://pod1-cai.mrel.infaqa.com/active-bpel/public/soap/6U0K7D0LiO5dD8PAnQDloq/HelloWorldProcess
"View WSDL File"

3. Click **View WSDL File** to open the complete WSDL and view the SOAP message generated for the process. You can copy the URL to use it as an endpoint.

   Note: If a process contains Japanese characters in its name, you might see an error when you view the WSDL file from a Chrome browser. If you see an error, right-click on the page and select **View page source** to view the WSDL file.

REST and Process Designer SOAP Endpoints to Access Message Events

You can access message events using REST and SOAP 1.1 endpoints generated by Process Designer. If SOAP 1.2 endpoints are required please use Process Developer to generate these endpoints. To obtain endpoints of services created using Process Developer, refer to information on policy-driven bindings in the *Process Developer Guide*.

The following table specifies the pattern for each REST or SOAP 1.1 endpoint based on the type of access required when you invoke message events.

The process uses HTTP Basic access authentication when you uncheck **Allow anonymous access** in the process Start properties or the Messages properties for message events. You can define a process to authenticate differently than the callback it receives, in message events or subprocess calls.

Type of Access	REST URL Format	SOAP 1.1 URL Format
Process with HTTP Basic access authentication	https://{host}/active-bpel/rt/{ProcessName}	https://{host}/active-bpel/soap/{ProcessName}
Process with OAuth2 authentication	https://{host}/active-bpel/rt/{ProcessName}	https://{host}/active-bpel/soap/{ProcessName}
Process with Anonymous access	https://{host}/active-bpel/public/rt/{TenantName}/{ProcessName}	https://{host}/active-bpel/public/soap{TenantName}/{ProcessName}
Process Message Event with Anonymous access	https://{host}/active-bpel/public/rt/{TenantName}/{ProcessName}/event/{MessageName}	https://{host}/active-bpel/public/soap/{TenantName}/{ProcessName}/event/{MessageName}
Process Message Event with HTTP Basic access authentication	https://{host}/active-bpel/rt/{ProcessName}/event/{MessageName}	https://{host}/active-bpel/soap/{ProcessName}/event/{MessageName}

Process Message Event in subProcess with Anonymous access	https://{host}/active-bpel/public/rt/{TenantName}/{SubProcessName}/event/{MessageName}	https://{host}/active-bpel/public/soap/{TenantName}/{ProcessName}/event/{MessageName}
		or
		https://{host}/active-bpel/public/soap/{TenantName}/{SubProcessName}/event/{MessageName}

Process Message Event in subProcess with HTTP Basic access authentication	https://{host}/active-bpel/rt/{SubProcessName}/event/{MessageName}	https://{host}/active-bpel/soap/{ProcessName}/event/{MessageName}
		or
		https://{host}/active-bpel/soap/{SubProcessName}/event/{MessageName}

Note: For SOAP requests, you can use the default process or sub process endpoint if the event has the same access, either anonymous or HTTP Basic access authentication, as the process or sub process.

Invoking Processes Deployed to the Secure Agent

If you use Secure Agent 30.0 or later, you can use HTTP and HTTPS endpoints to invoke processes deployed to the Secure Agent.

Use the following sections to construct or access REST and SOAP endpoints to access processes deployed to the Secure Agent through HTTP or HTTPS. When you deploy Informatica Process Designer (IPD) processes, the endpoint URL construction for anonymous access is different from the endpoint URL construction for non anonymous access.

Invoking Process Designer Processes

Type of Access	REST Endpoint	SOAP 1.2 Endpoint
Non anonymous access of an IPD process using HTTP.	http://[host][:port]/process-engine/rt/[serviceName] Swagger: http://[host][:port]/process-engine/rt/[serviceName]?swagger	http://[host][:port]/process-engine/soap/[serviceName] WSDL: http://[host][:port]/process-engine/soap/[serviceName]?wsdl
Anonymous access of an IPD process using HTTP.	http://[host][:port]/process-engine/public/rt/[serviceName] Swagger: http://[host][:port]/process-engine/public/rt/[serviceName]?swagger	http://[host][:port]/process-engine/public/soap/[serviceName] WSDL: http://[host][:port]/process-engine/public/soap/[serviceName]?wsdl
Non anonymous access of an IPD process using HTTPS.	https://[host][:port]/process-engine/rt/[serviceName] Swagger: https://[host][:port]/process-engine/rt/[serviceName]?swagger	https://[host][:port]/process-engine/soap/[serviceName] WSDL: https://[host][:port]/process-engine/soap/[serviceName]?wsdl
Anonymous access of an IPD process using HTTPS.	https://[host][:port]/process-engine/public/rt/[serviceName] Swagger: https://[host][:port]/process-engine/public/rt/[serviceName]?swagger	https://[host][:port]/process-engine/public/soap/[serviceName] WSDL: https://[host][:port]/process-engine/public/soap/[serviceName]?wsdl

Invoking Process Developer Processes

Perform the following steps to get a list of Process Developer SOAP and REST endpoint URLs that you can invoke:

1. Open the **Administrative Services** page at
   - `https://localhost:7443/process-engine/` for HTTPS endpoint URLs.
   - `http://localhost:7080/process-engine/` for HTTP endpoint URLs.
2. Click **Visit the Web Services Listing** or **Visit the REST Service**.
3. Enter the user credentials of an Informatica Cloud Services administrator.

You see a list of SOAP or REST endpoint URLs. Use a SOAP or REST client to invoke these endpoint URLs.

Invoking Processes Deployed to Secure Agent Groups

You can invoke a process that you deploy to a specific Secure Agent or to a Secure Agent group through a SOAP/REST URL. If you deploy a process to a Secure Agent group, you can also invoke the process through a load balancing URL.

You can invoke a process in the following ways:

Invoke a Process Deployed to a Specific Secure Agent

You can deploy a process to a specific Secure Agent within a Secure Agent group. To invoke such a process, use the SOAP/REST service URLs on the **Design Home** page. The SOAP/REST service URLs reflect the properties of the Secure Agent you deployed the process to.

Invoke a Process Deployed to a Secure Agent Group

You can deploy a process to a Secure Agent group. To invoke such a process, use the SOAP/REST service URLs on the **Design Home** page. The SOAP/REST URLs reflect the properties of the master Secure Agent of the Secure Agent group.

If you deploy a process to a Secure Agent group and you have a load balancer, you can create a load balancer URL. You can invoke the process through the load balancer URL in addition to the SOAP/REST service URLs on the **Design Home** page.

To learn more about Secure Agent groups, see *Secure Agent Services* in the *Administrator* help.

Invoking processes using HTTP verbs

You can use the following HTTP verbs to invoke a process:

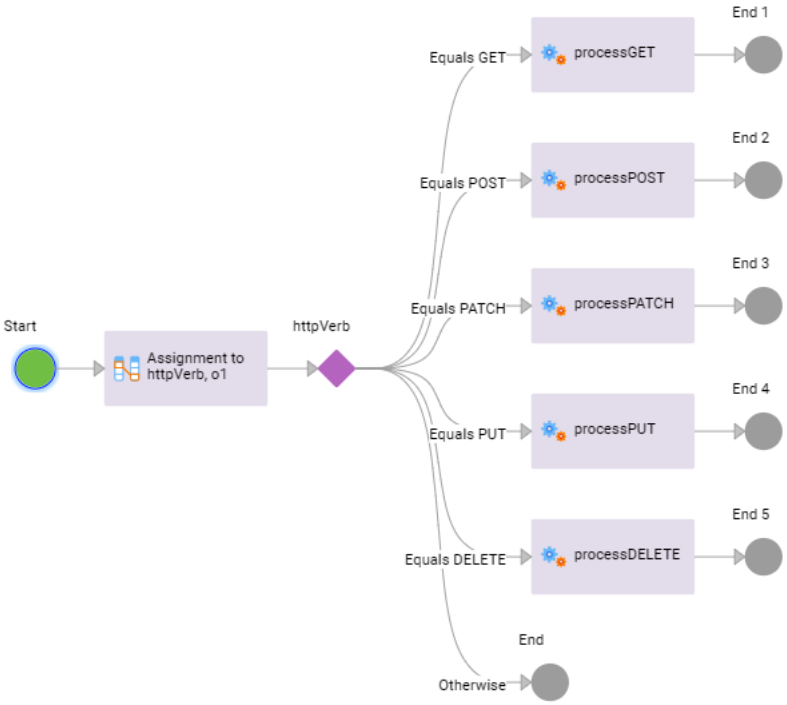
- GET
- POST
- PATCH
- PUT
- DELETE

You can configure a single process to use different HTTP verbs and perform different CRUD operations based on the HTTP verb that is used in a request. You can use XQuery functions to determine the HTTP verb and the resource path segments that are used in a request.

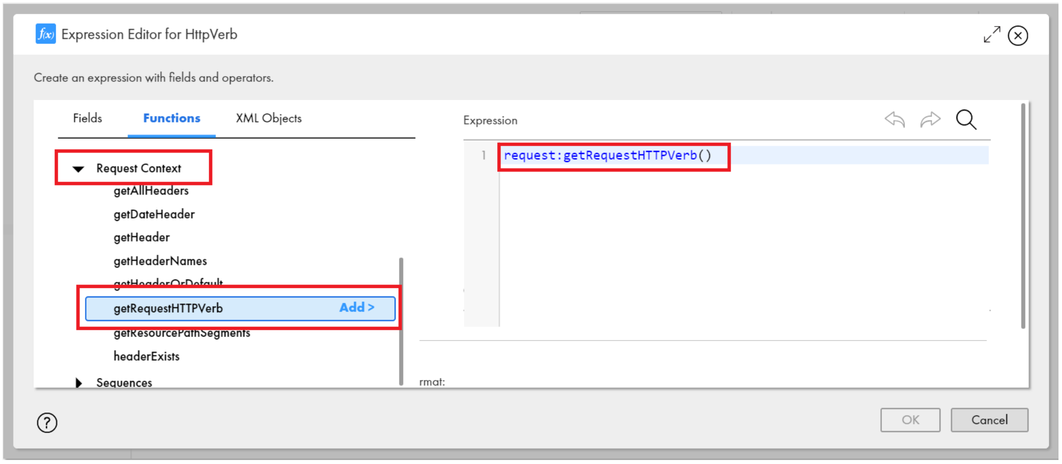
For example, you want to update a CRM system with customer records. You might want to perform one or more of the following tasks:

- Use the GET verb to read a customer record.
- Use the POST verb to insert a new customer record.
- Use the PATCH verb to update an existing customer record.
- Use the PUT verb to overwrite an existing customer record.
- Use the DELETE verb to delete an existing customer record.

You can configure a single process to perform all these operations as shown in the following image:



The Assignment step in the process uses the `getRequestHTTPVerb` function to determine the HTTP verb that is used in the request. The following image shows the `getRequestHTTPVerb` function under the **Request Context** section in the Expression Editor:



Expression Editor for HttpVerb

Create an expression with fields and operators.

```
request:getRequestHTTPVerb()
```

Based on the HTTP verb that is used, the Decision step branches out to different paths to perform different CRUD operations.

To view a video about using different HTTP verbs to invoke a process, see the following community article:

https://www.youtube.com/watch?v=bQaQ_KmBO4o

HTTP verb functions

Use HTTP verb functions to retrieve the HTTP verb and the resource path segments that are used in a request. The HTTP verb functions are available under the **Request Context** section of the Expression Editor.

You can use the following HTTP verb functions:

getRequestHTTPVerb

Use the `getRequestHTTPVerb` function to determine the HTTP verb that is used in a request. The function retrieves the HTTP verb from the request and returns one of the following responses in a string format:

- GET
- POST
- PATCH
- PUT
- DELETE

getResourcePathSegments

Use the `getResourcePathSegments` function to retrieve all or specific resource path segments of REST requests. The function returns the values as a string of tokens.

For example, consider the following request URL:

```
https://na1.ai.dm-us.informaticacloud.com/active-bpel/rt/InitiateOrder/Orders/
OrderID_100/quantity/5
```

If you use the `request:getResourcePathSegments()` expression, you see the following response:

```
[Orders, OrderID_100, quantity, 5]
```

To retrieve a specific resource path segment, use a numeric qualifier to denote the position of the resource path segment from the process name.

For example, to retrieve the `OrderID_100` segment alone in the response, use the following expression:

```
request:getResourcePathSegments()[2]
```

Note: You cannot use the `getResourcePathSegments` function to fetch the resource path segments of SOAP requests and message events. You cannot use the `getResourcePathSegments` function to fetch the host context.

REST Endpoints and Data Conversion

REST Endpoints

You can expose a process as a REST service through a URL that uses the following format:

```
https://mySPIwebAdress:portNumber/active-bpel/services/REST/<process_name>?var=value
```

You can use one of the following verbs to invoke a process:

- GET
- POST

- PATCH
- PUT
- DELETE

Use JSON when you send a request with a POST verb.

For example, the following snippet shows a JSON request that is used with a POST verb:

```
{
        "param_A" : "abc", // NOTE: string, no $t
        "param_B" : 123,   // NOTE: typed as a number
        "param_C" : true   // NOTE: typed as bool
 }
```

The request returns the following response:

```
{
        "field_A" : "abc",
        "field_B" : 123,
        "field_C" : true
 }
```

Use query parameters when you send a request with the GET verb.

For example, the following snippet shows the same request with a GET verb:

```
GET/rt/ProcessName?param_A=abc&param_B=123&param_C=true
```

Simple Array Handling

You can place a JSON payload that contains an array of simple types in a process object. For example, in a process, you can define an input field named `orders` of type `ObjectList`. In the `orders` input field, you can reference a process object named `OrderNumber` that contains a single field.

The following snippet shows the JSON array that can be sent:

```
{
 "orders" : ["O-213", "O-425"]
}
```

Process Designer transforms this internally into:

```
{
 "orders" : [{OrderNumber : "O-213"},
             {OrderNumber : "O-425"}]
}
```

When an output field is serialized to JSON, Process Designer transforms the objects into an array of simple types.

Swagger JSON specification of a process

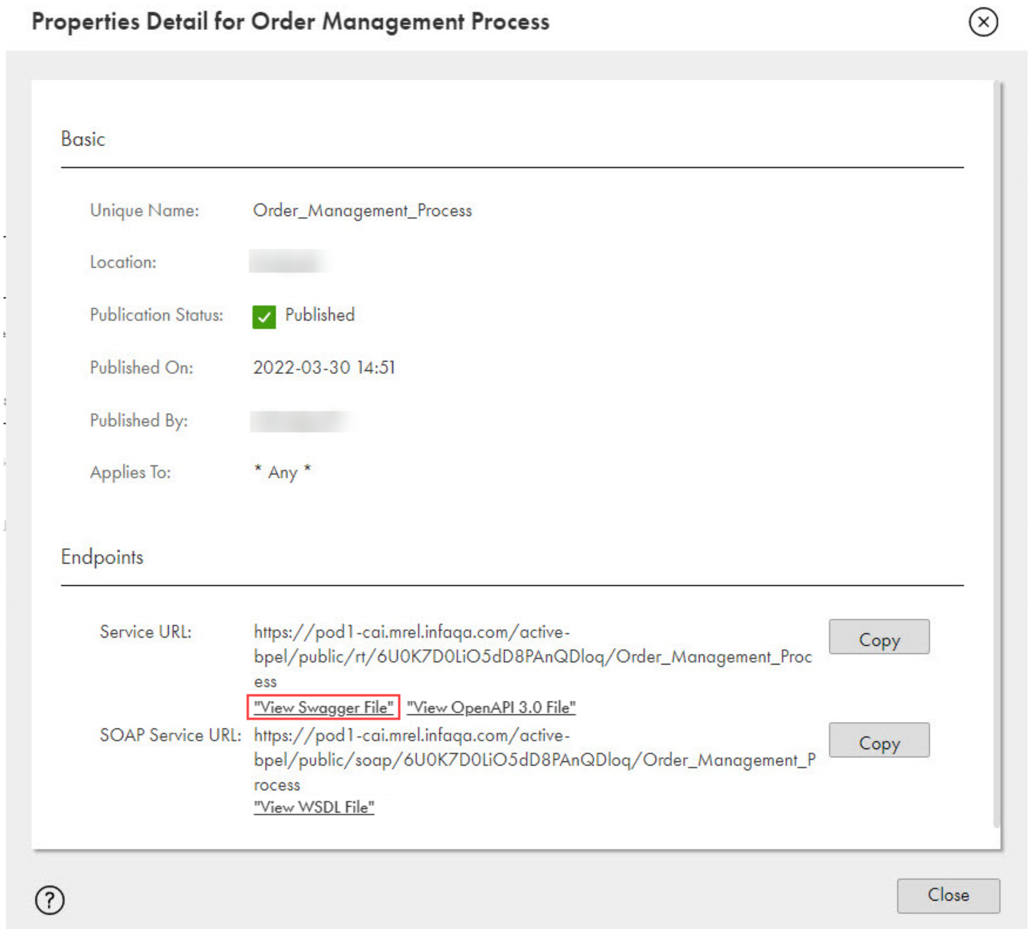
You can view the Swagger JSON specification of the payload of a process that you create in Informatica Process Designer.

The Swagger JSON specification of a process contains process payload features such as input, output, and process objects. The Swagger JSON specification contains the standard Swagger property requirements of `version`, `info`, `host`, `basePath`, `schemes`, and `paths`.

Accessing the Swagger JSON specification of a process

To access the Swagger JSON description of a process, perform one of the following tasks:

- Open the process for which you want to view the Swagger JSON specification and click **Actions** > **Properties Detail**. Then, click **View Swagger File** next to the service URL.
  The following image shows the link to access the Swagger file:



Properties Detail for Order Management Process

Basic

Unique Name: Order_Management_Process

Location: [redacted]

Publication Status: ✓ Published

Published On: 2022-03-30 14:51

Published By: [redacted]

Applies To: * Any *

Endpoints

Service URL: https://pod1-cai.mrel.infaqa.com/active-bpel/public/rt/6U0K7D0LiO5dD8PAnQDloq/Order_Management_Process

"View Swagger File" "View OpenAPI 3.0 File"

SOAP Service URL: https://pod1-cai.mrel.infaqa.com/active-bpel/public/soap/6U0K7D0LiO5dD8PAnQDloq/Order_Management_Process

"View WSDL File"

- Open the process and click **Actions** > **Properties Detail** > **Copy** next to the service URL. Then, in a browser window, paste the service URL and append `?Swagger` to the service URL.
  For example, if the service URL is `https://bs1e1.rt.informaticacloud.com/active-bpel/rt/sample_process`, paste `https://bs1e1.rt.informaticacloud.com/active-bpel/rt/sample_process?Swagger` in the browser window.

The Swagger JSON specification for the processes shows the HTTP response status code, step name, and the HTTP response header name and type that you configured in the End step or the Milestone step. However, if a process contains a Receive step, the HTTP response status code remains **200 OK** irrespective of the configuration. For more information about the response status codes that Application Integration supports, see Appendix A, “HTTP response status codes” on page 35.

Using the Swagger JSON specification of a process

You can use the Swagger JSON specification of a process with third-party tools. For example, you can use a Swagger JSON specification of a process with the Swagger.io codegen tool, http://swagger.io/swagger-codegen/.

You can use the Swagger JSON description with internal tools. For example, you can use the Swagger JSON specification of a process to create a file that you use to import a service connector. For more details, see *Design > Creating Service Connectors > Generating Service Connectors*.

OpenAPI JSON specification of a process

The OpenAPI specification, previously called Swagger specification, is a format for describing REST API endpoints.

Application Integration supports OpenAPI specification version 3.0.1. You can view the OpenAPI specification of the payload of a process that you create in Process Designer.

The OpenAPI JSON specification of a process contains process payload information such as input, output, descriptions, and process objects. The OpenAPI specification specifies the standard OpenAPI property requirements of `version`, `info`, `servers`, `paths`, and `components`.

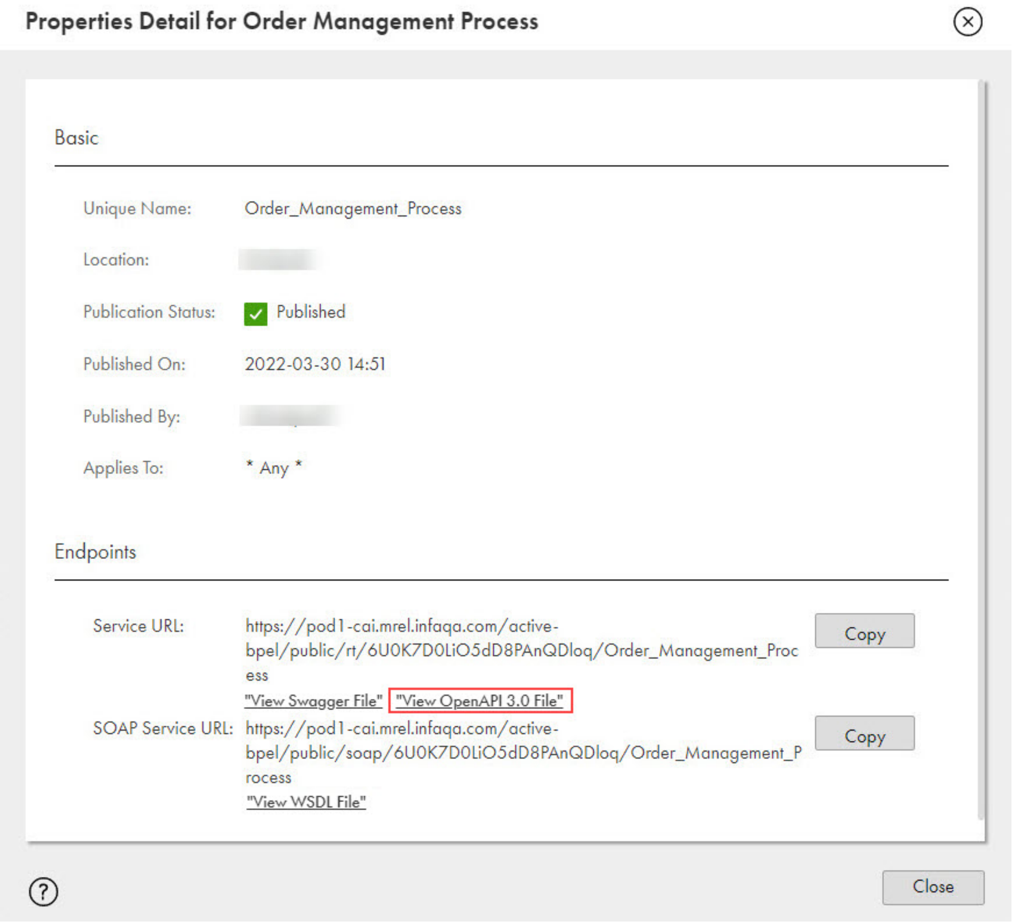
The OpenAPI 3.0.1 specification is more descriptive than Swagger 2.0 and allows you to create complex documents.

Accessing the OpenAPI JSON specification of a process

To access the OpenAPI 3.0 specification of a process, perform the following steps:

1. On the **Explore** page, navigate to the process for which you want to view the OpenAPI specification.
2. On the **Actions** menu, click **Properties Detail** to view the generated OpenAPI file.
   The **Properties Detail** dialog box appears.

   The following image shows the link to access the OpenAPI 3.0 file:



Properties Detail for Order Management Process

Basic

Unique Name: Order_Management_Process

Location: [redacted]

Publication Status: ✓ Published

Published On: 2022-03-30 14:51

Published By: [redacted]

Applies To: * Any *

Endpoints

Service URL: https://pod1-cai.mrel.infaqa.com/active-bpel/public/rt/6U0K7D0LiO5dD8PAnQDloq/Order_Management_Process

"View Swagger File" "View OpenAPI 3.0 File"

SOAP Service URL: https://pod1-cai.mrel.infaqa.com/active-bpel/public/soap/6U0K7D0LiO5dD8PAnQDloq/Order_Management_Process

"View WSDL File"

3. Click **View OpenAPI 3.0 File** to view the associated OpenAPI file.
   The OpenAPI 3.0.1 specification opens on a new tab.

The OpenAPI 3.0.1 specification for the processes shows the HTTP response status code, step name, and the HTTP response header name and type that you configured in the End step or the Milestone step. However, if a process contains a Receive step, the HTTP response status code remains **200 OK** irrespective of the configuration. For more information about the response status codes that Application Integration supports, see Appendix A, “HTTP response status codes” on page 35.

Running a process

After you publish a process, you can run a process to test it. You can create process inputs and run the process with process inputs. After you run the process, you can view details of the successful and unsuccessful executions of the process instances.

The following video shows you how to create process inputs and test a process with process inputs:

https://knowledge.informatica.com/s/article/DOC-18509

Creating a process input

After you publish a process, you can create a process input and use it to run a process for testing purposes. You can also create multiple process inputs and run a process with all the inputs.

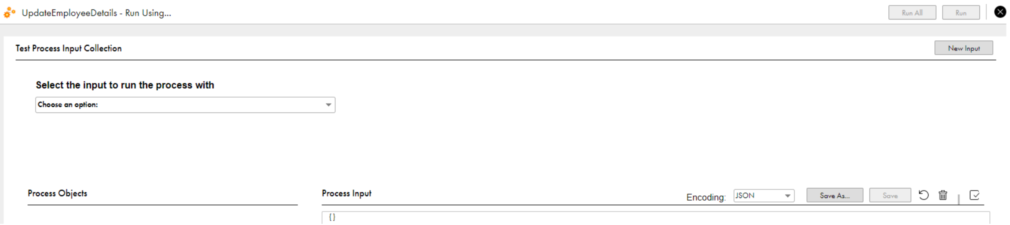
You can create a process input in the JSON or XML format. Then, validate and save the process input.

When you export, copy, or move a process that contains process inputs, the process inputs are also retained.

1. On the **Explore** page, navigate to the process for which you want to create a process input.
2. On the **Actions** menu, click **Run Using**.

   The **Test Process Input Collection** page opens.

   The following image shows the **Test Process Input Collection** page:



UpdateEmployeeDetails - Run Using...
Run All
Run
Test Process Input Collection
New Input
Process Objects
Process Input
Save As...
Save

Select the input to run the process with: Choose an option:

Encoding: JSON

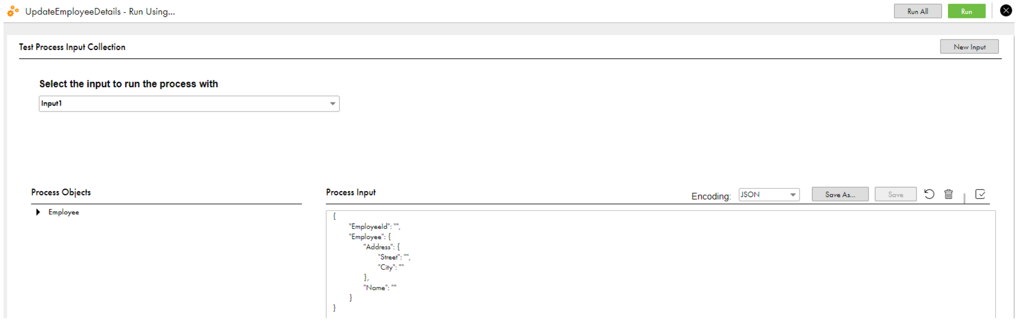
```
{}
```

3. Click **New Input**.
4. Enter a name for the process input and click **Save**.

   The name of the process input must not exceed 80 characters.

   The **Process Input** section is populated with the input fields required for the process.

   The following image shows the **Process Input** section populated with input fields for an **Employee** process object:



Select the input to run the process with: Input1

Encoding: JSON

```
{
    "EmployeeId": "",
    "Employee": {
        "Address": {
            "Street": "",
            "City": ""
        },
        "Name": ""
    }
}
```

5. From the **Encoding** list, select **JSON** or **XML** based on the format that you want to work with and specify the process input.

   Note: If the process contains input fields with space characters in them, select **JSON** as the encoding type.
6. Click the **Validate** icon to validate the syntax of the process input.

   A confirmation message appears stating if the validation was successful or not. If the validation fails, correct the syntax of the process input, and validate again.
7. Click **Save** to save the process input.

   You can click **Save As** to save the process input with a different name. You can also click the **Reset** icon reset a process input to the last saved process input.

After you create process inputs, you can run the process with the required inputs.

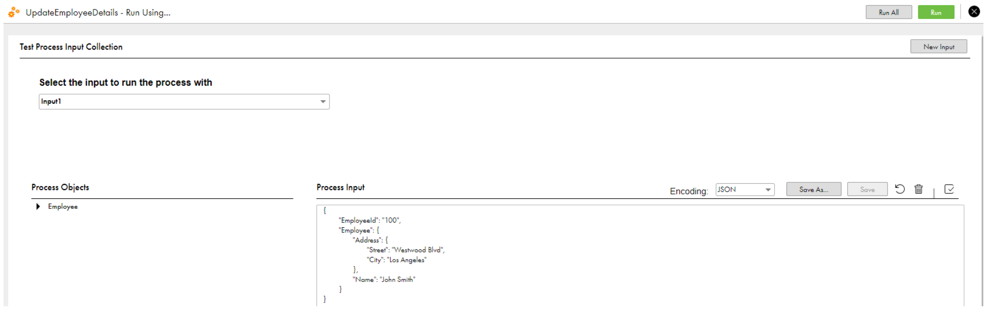
Running a process with process inputs

After you publish a process and create process inputs, you can run the process with process inputs to test it. After you run the process, you can view details of the successful and unsuccessful executions of the process instances.

1. On the **Explore** page, navigate to the process that you want to run with one or more process inputs.
2. Perform one of the following steps:
    - To run a process with the last successfully run process input, open the **Actions** menu, and click **Run**. The **Process Execution Status** page opens displaying details of the successful and unsuccessful process executions.

      Note: If you had not created process inputs or run the process before, the **Test Process Input Collection** page opens from where you can create process inputs and run the process.
    - To run a process with a specific process input or all the process inputs, open the **Actions** menu and click **Run Using**. The **Test Process Input Collection** page opens.

      The following image shows the **Test Process Input Collection** page:



Select the input to run the process with: Input1

Encoding: JSON

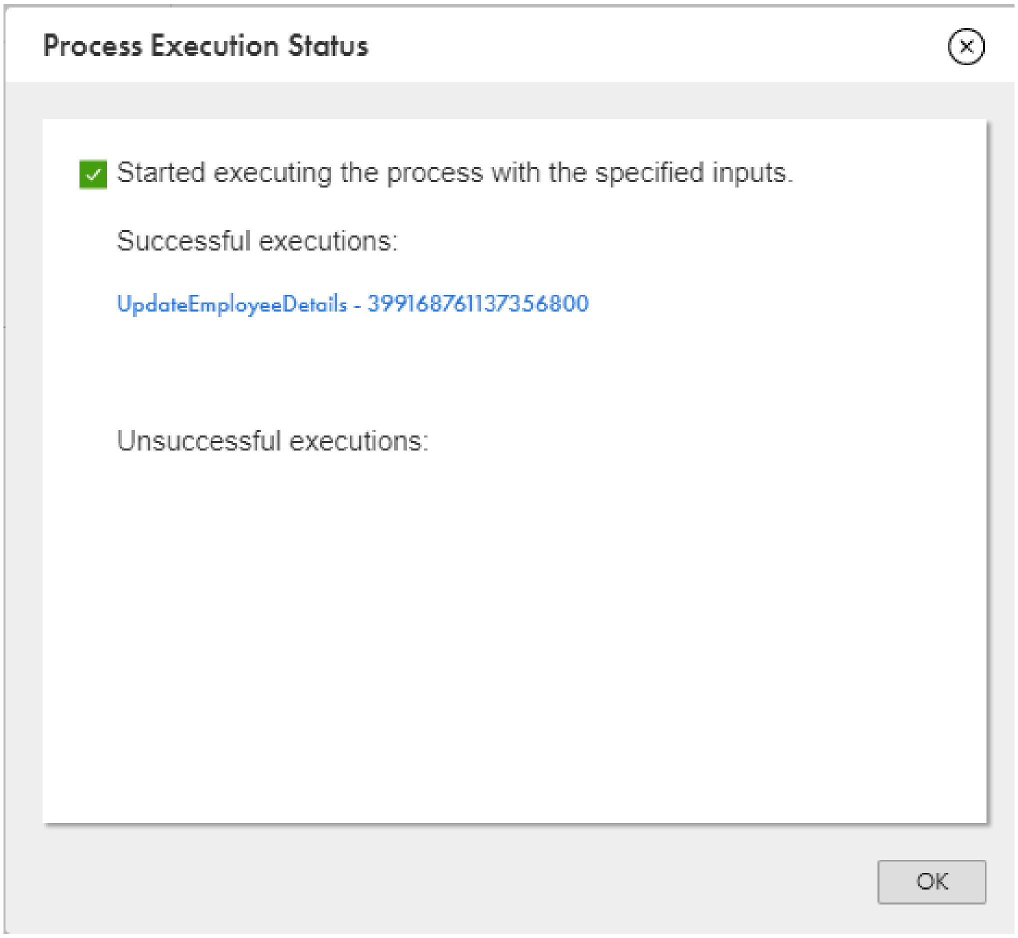
```
{
    "EmployeeId": "100",
    "Employee": {
        "Address": {
            "Street": "Westwood Blvd",
            "City": "Los Angeles"
        },
        "Name": "John Smith"
    }
}
```

   Note: The **Run** and **Run Using** options are disabled if the process contains unsaved or unpublished changes. To run the process, you must save and publish the process again.

3. Perform one of the following steps:
    - To run the process with a specific process input, select the process input and click **Run**.
    - To run the process with all the process inputs, click **Run All**.

   Application Integration runs the process with the specified inputs. After the process execution is complete, the **Process Execution Status** page opens displaying details of the successful and unsuccessful process executions. For long running processes, it might take some time for the **Process Execution Status** page to appear.

   The following image shows the **Process Execution Status** page:



✓ Started executing the process with the specified inputs.

Successful executions:

UpdateEmployeeDetails - 399168761137356800

Unsuccessful executions:

4. Click the links under the **Successful executions** or **Unsuccessful executions** sections to view details of the process execution.

   The process run ID is appended to the link. You can use the run ID to verify the process execution details in the Application Integration Console.

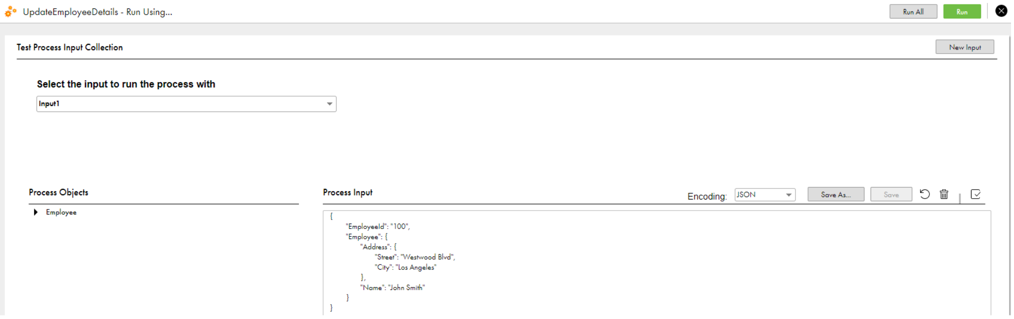
Deleting a process input

You can delete a process input if you do not need it.

1. On the **Explore** page, navigate to the process for which you want to delete a process input.
2. On the **Actions** menu, click **Run Using**.

   The **Test Process Input Collection** page opens.

   The following image shows the **Test Process Input Collection** page:



Select the input to run the process with: Input1

Encoding: JSON

```
{
    "EmployeeId": "100",
    "Employee": {
        "Address": {
            "Street": "Westwood Blvd",
            "City": "Los Angeles"
        },
        "Name": "John Smith"
    }
}
```

3. Select the process input that you want to delete, and click the **Delete** icon.

   A message appears prompting you to confirm the process input deletion.

4. Click **Delete** to proceed with the deletion or click **Cancel** to cancel the deletion.

Rules and guidelines for process inputs

Consider the following rules and guidelines when you create process inputs to run a process:

- If a process contains input fields with space characters in them, you cannot use the XML encoding type while creating the process input.
- You cannot create a process input of the Image, Attachment, and Attachments data types.
- If you create a process input of the Rich Text Area data type, you must verify that the process input is properly encoded. You must encode `< >` as `&lt; &gt;`.
  Consider the name of the text input field as `InputField` and the text input value as `HelloWorld` in bold font. For this example, you must edit the process input code as follows:

  ```
  { "InputField": " &lt;bold&gt; HelloWorld &lt;/bold&gt; " }
  ```

- If you create a process input of the Any data type, you must specify the process input as a key value pair. Consider the name of the input field of the Any data type as `InputFieldAnyType` and the value as `valueHelloWorld`. For this example, the process input code with JSON encoding must be as follows:

  ```
       {{ "InputFieldText": "HelloWorld", "InputFieldAnyType": { "key": "valueHelloWorld" }}
  ```

  If you select the encoding type as **XML**, the process input code must be as follows:

  ```
      <root>
      <InputFieldText>HelloWorld</InputFieldText>
      <InputFieldAnyType>
      <key>HelloWorld</key>
      </InputFieldAnyType>
      </root>
  ```

- In the **Process Objects** section, only the input fields that contain process objects appear.
- When you make changes to the input fields in the published process, you must manually refresh the **Run Using** dialog box to see the changes.
- When you make changes to the process input and cancel the changes without saving them, the changes persist.
- If a process contains an output field of the list of number data type with a precision greater than 7, the value is displayed in the exponential format for JSON payloads in the advanced view of a process in Application Integration Console.

Activity execution limit restriction for process invocations

If the process invocation fails due to an activity limit breach of 10,000, you cannot run the process for the next 15 minutes and no new instance of the process deployment will be created during this period.

When you run a process that does not have an appropriate retry or loop breaker mechanism, the following error occurs:

```
{"error":{"code":429,"message":"The last execution of this process has exceeded the
activity execution limit at [07-Sep-2023 06:55:23 GMT]. As a result, new instances of
this process deployment will not be created for a period of [15] minutes. You can review
the process logs to find out the root cause of the activity limit breach."}}
```

You can review the process logs to find out the root cause of the activity limit breach.

You can resume the process execution after the 15-minute time limit expires or by republishing the process.

When you publish a process that goes into an infinite loop on a Secure Agent cluster, you might encounter a conflict in the behavior and not see the error message every time you run the process. This is because the in-memory cache used to store the process instances is node-specific and not broadcasted to all the nodes in the Secure Agent cluster. The system automatically chooses the nodes, and in a multi-node cluster setup, there might be cases where a process is rejected on one node but executed on another node. Each node maintains its own cache, allowing for independent handling of process failures and potential retries depending on the node where the process is executed.

Example

Process A during its execution on node 1 encounters a `429- Too Many Requests` error. As a result, an entry is added to the in-memory cache specific to node 1, indicating that process A failed due to an activity limit breach. The new instances of process A on node 1 will not be created for a period of 15 minutes.

However, if the process A execution is routed to node 2, node 2 will attempt to execute the process again. If process A encounters the same `429- Too Many Requests` error on node 2, a new entry will be added to the cache specific to node 2.

Note: The activity execution limit restriction for process invocations feature is in technical preview. Technical preview functionality is supported but is unwarranted and is not production-ready. Informatica recommends that you use in non-production environments only. Informatica intends to include the preview functionality in an upcoming GA release for production use, but might choose not to in accordance with changing market or technical circumstances. For more information, contact Informatica Global Customer Support.

Process termination

If a non-persistent process runs for longer than 3600 seconds, that is, 1 hour, it will automatically time out and get terminated. The timeout ensures that system resources are not blocked.

Process retention

Processes follow the retention (purge) policy set by Informatica.

If automated database maintenance is enabled, the default retention days set for all processes are applied to the process. By default, a completed process instance remains in the Process Server database for 24 hours, that is, 1 day. A faulted process instance remains in the Process Server database for 72 hours, that is, 3 days. The process instances are automatically deleted after the retention period expires. The server logs remain in the Process Server database for 30 days.

The process instance retention setting configured for an individual process in Application Integration Console takes precedence over the default retention days setting for all processes.

Note: The retention period information in this topic might change without prior notice.

For more information about process instance retention, see Process Instance Retention.

CHAPTER 2

Runtime tasks for guides

After you create a guide, you can perform the following runtime tasks:

Publish the guide

You must publish a guide to run the guide or embed the guide within a third-party application.

Run the guide

After you publish a guide, you can run the guide from Guide Designer to test the guide.

Embed the guide within a third-party application

After you publish a guide, Application Integration generates an embed code. You can use the embed code to embed the guide within a third-party application.

Publishing a guide

You must publish a guide to run the guide or embed the guide within a third-party application.

You can also publish multiple guides in bulk. For more information, see Chapter 3, "Publishing Application Integration assets in bulk" on page 28.

If you edit a published guide, you must publish the guide again for the changes to get reflected. Otherwise, the guide status changes to **Outdated** in the **Property Details** dialog box and on the **Explore** page. The **Outdated** status indicates that the guide contains unpublished changes.

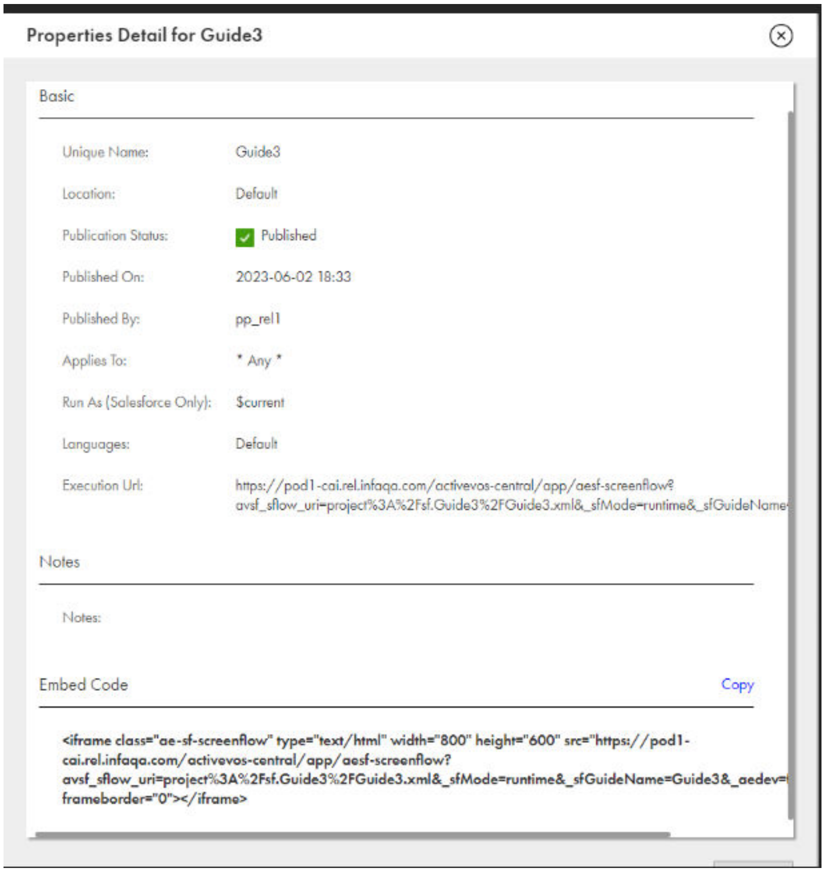
1. On the **Explore** page, navigate to the guide that you want to publish and click **Publish**.

   Application Integration publishes the guide, and generates an execution URL and embed code.

2. On the **Actions** menu, click **Properties Detail** to view the generated execution URL and embed code.

   The **Properties Detail** dialog box appears.

   The following image shows the **Properties Detail** dialog box:



Properties Detail for Guide3

Basic

Unique Name:	Guide3
Location:	Default
Publication Status:	☑ Published
Published On:	2023-06-02 18:33
Published By:	pp_rel1
Applies To:	* Any *
Run As (Salesforce Only):	$current
Languages:	Default
Execution Url:	https://pod1-cai.rel.infaqa.com/activevos-central/app/aesf-screenflow?avsf_sflow_uri=project%3A%2Fsf.Guide3%2FGuide3.xml&_sfMode=runtime&_sfGuideName

Notes

Notes:

Embed Code

```
<iframe class="ae-sf-screenflow" type="text/html" width="800" height="600" src="https://pod1-
cai.rel.infaqa.com/activevos-central/app/aesf-screenflow?
avsf_sflow_uri=project%3A%2Fsf.Guide3%2FGuide3.xml&_sfMode=runtime&_sfGuideName=Guide3&_aedev=
frameborder="0"></iframe>
```

3. In the **Execution URL** field, you can view the execution URL that you can use to run the guide.
4. In the **Embed Code** section, click **Copy** to copy the embed code.

   You can then use the embed code to embed the guide within a third-party application.

   Note: If the third-party application doesn't support the embed code, you can use the execution URL to run the guide.

Unpublishing a guide

To edit the guide name or disable a published guide, you must unpublish the guide. You cannot invoke an unpublished guide from a third-party application. Therefore, after you unpublish a guide, you must update the third-party application accordingly.

1. On the **Explore** page, navigate to the guide that you want to unpublish.
2. On the **Actions** menu, click **Unpublish**.

   Application Integration unpublishes the guide, and disables the guide and the embed code.

After you make the necessary changes, publish the guide again to generate the embed code and enable the guide.

Running a guide

After you publish a guide, you can run the guide to test it.

If the guide applies to a Salesforce object, run the guide in Salesforce. For more information about running a guide in Salesforce, see *Salesforce and Application Integration Guide*.

Perform the following steps to run a guide from Guide Designer:

1. On the **Explore** page, navigate to the guide that you want to run.
2. On the **Actions** menu, click **Run**.

   Note: The **Run** option is disabled if the guide contains unsaved or unpublished changes. To run the guide, you must save and publish the guide again.

   The guide opens in a browser.

Embedding a guide within a third-party application

When you publish a guide, Application Integration generates an embed code for the guide. The embed code contains an <iframe> tag, which specifies an inline frame. You can use the inline frame to embed the guide into an HTML document of a third-party application.

Guide termination

If a user does not provide an input for a guide and leaves it running, the guide becomes inactive. Inactive guides are automatically terminated after 7 days. If an inactive guide is at a Milestone step, the guide is automatically terminated after 14 days.

Guide retention

Guides follow the retention (purge) policy set by Informatica.

If automated database maintenance is enabled, the default retention days set for all guides are applied to the guides. By default, a completed guide instance remains in the Process Server database for 24 hours, that is, 1 day. A faulted guide instance remains in the Process Server database for 72 hours, that is, 3 days. The guide instances are automatically deleted after the retention period expires. The server logs remain in the Process Server database for 30 days.

Note: The retention period information in this topic might change without prior notice.

CHAPTER 3

Publishing Application Integration assets in bulk

You can use the publish resource to publish a single Application Integration asset or multiple Application Integration assets simultaneously and save time. You can publish processes, guides, app connections, and service connectors in bulk.

The assets are published in the same order as given in the request payload. You can publish a maximum of 199 assets at a time.

1. In a REST client, use a POST request with the following URL:

   `<Cloud Application Integration POD URL>/active-bpel/asset/v1/publish`

2. Add the following headers:

Key	Value
Accept	application/vnd.api+json
Content-Type	application/vnd.api+json
INFA-SESSION-ID	Use the login resource to get the session ID. For more information about the login resource, see *REST API Reference* in the Data Integration help.

3. In the body, use the assetPaths attribute to specify one or more locations and names of the Application Integration assets that you want to publish.

   Use the following format:

```
{

  "data": {
    "type": "publish",
    "attributes": {
      "assetPaths": [
                        "Explore/<location-of-the-process>/<name-of-the-
process>.PROCESS.xml",
                        "Explore/<location-of-the-guide>/<name-of-the-
guide>.GUIDE.xml",
                        "Explore/<location-of-the-appconnection>/<name-of-the-
appconnection>.AI_CONNECTION.xml",

           "Explore/<location-of-the-serviceconnector>/<name-of-the-
serviceconnector>.AI_SERVICE_CONNECTOR.xml"
                    ]
    }
  }
}
```

4. Send the POST request.

   You see a publish job ID and a success or failure response. If the request fails, the response also gives the error details.

   The following snippet shows a sample response:

```
{
    "data": {
        "type": "publish",
        "id": "690465325825028096",
        "attributes": {
            "jobState": "NOT_STARTED",
            "jobStatusDetail": {},
            "startedBy": "autouser_pod1",
            "startDate": "2022-03-21T07:42:42.000+0000",
            "totalCount": 1,
            "processedCount": 0,
            "assetPaths": [
                "Explore/Default/BulkPublishProcessToPublishOnCloud.PROCESS.xml"
            ]
        }
    },
    "links": {
        "self": https://na1.ai.dm-us.informaticacloud.com/active-bpel/asset/v1/
publish/690465325825028096,
        "status": https://na1.ai.dm-us.informaticacloud.com/active-bpel/asset/v1/
publish/690465325825028096/Status
    }
}
```

   The publish job ID in this example is 690465325825028096.

5. To view information about the publish status and publish job, use a GET request with the following URLs:

URL	Description
`<Cloud Application Integration POD URL>/active-bpel/asset/v1/publish/<publishId>/Status`	Displays the publish status.
`<Cloud Application Integration POD URL>/active-bpel/asset/v1/publish/<publishId>`	Displays the publish information.

CHAPTER 4

Unpublishing Application Integration assets in bulk

You can use the unpublish resource to unpublish a single Application Integration asset or multiple Application Integration assets simultaneously and save time. You can unpublish processes, guides, app connections, and service connectors in bulk.

The assets are unpublished in the same order as given in the request payload. You can unpublish a maximum of 199 assets at a time.

1. In a REST client, use a POST request with the following URL:

   `<Cloud Application Integration POD URL>/active-bpel/asset/v1/unpublish`

   For example: `https://na1.dm-us.informaticacloud.com/active-bpel/asset/v1/unpublish`
2. Add the following headers:

Key	Value
Accept	application/vnd.api+json
Content-Type	application/vnd.api+json
INFA-SESSION-ID	Use the login resource to get the session ID. For more information about the login resource, see *REST API Reference* in the Data Integration help.

3. In the body, use the assetPaths attribute to specify one or more locations and names of the Application Integration assets that you want to unpublish.

   Use the following format:

```
{

  "data": {
    "type": "unpublish",
    "attributes": {
      "assetPaths": [
                        "Explore/<location-of-the-process>/<name-of-the-
process>.PROCESS.xml",
                        "Explore/<location-of-the-guide>/<name-of-the-
guide>.GUIDE.xml",
                        "Explore/<location-of-the-appconnection>/<name-of-the-
appconnection>.AI_CONNECTION.xml",

           "Explore/<location-of-the-serviceconnector>/<name-of-the-
serviceconnector>.AI_SERVICE_CONNECTOR.xml"
                    ]
```

```
        }
      }
    }
```

4. Send the POST request.

   You see an unpublish job ID and a success or failure response. If the request fails, the response also gives the error details.

   The following snippet shows a sample response:

```
{
    "data": {
        "type": "unpublish",
        "id": "547654765345095786",
        "attributes": {
            "jobState": "NOT_STARTED",
            "jobStatusDetail": {},
            "startedBy": "autouser_pod1",
            "startDate": "2022-03-21T07:42:42.000+0000",
            "totalCount": 1,
            "processedCount": 0,
            "assetPaths": [
                "Explore/Default/BulkUnpublishProcessToUnpublishOnCloud.PROCESS.xml"
            ]
        }
    },
    "links": {
        "self": https://na1.ai.dm-us.informaticacloud.com/active-bpel/asset/v1/
unpublish/547654765345095786,
        "status": https://na1.ai.dm-us.informaticacloud.com/active-bpel/asset/v1/
unpublish/547654765345095786/Status
    }
}
```

   The unpublish job ID in this example is 547654765345095786.

5. To view information about the unpublish status and unpublish job, use a GET request with the following URLs:

URL	Description
`<Cloud Application Integration POD URL>/active-bpel/asset/v1/unpublish/<unpublishId>/Status`	Displays the unpublish status.
`<Cloud Application Integration POD URL>/active-bpel/asset/v1/unpublish/<unpublishId>`	Displays the unpublish information.

Unpublishing dependent assets

When you have dependent assets, unpublishing them simultaneously might result in issues. You must first unpublish the dependent assets, and then send the bulk unpublish request.

For example, you have an app connection that uses a service connector, and both the assets are published. When you send a bulk unpublish request for the service connector, in the status response, the overall `jobState` is set to `ERROR`, and the `itemState` and `itemStatusDetail` provide the error details. To resolve this issue, you must first unpublish the dependent assets and send the bulk unpublish request again.

Note: If the `itemState` for an asset is `NOT_FOUND`, the `jobState` is set to `WARNING`. If the `itemState` for an asset is `ERROR`, the `jobState` is set to `ERROR`. If the `itemState` in the response contains both the `ERROR` and `NOT_FOUND` values, the `jobState` is set to `ERROR`.

The following snippet shows a sample response:

```
{
    "data": {
        "type": "unpublishStatus",
        "id": "937604908948291875",
        "attributes": {
            "jobState": "ERROR",
            "jobStatusDetail": {
                "itemStateSummary": {
                    "NOT_FOUND": 1,
                    "ERROR": 1,
                    "SUCCESS": 1
                }
            },
            "startedBy": "user_na1",
            "startDate": "2024-02-01T07:07:10.000+0000",
            "endDate": "2024-02-01T07:07:15.000+0000",
            "totalCount": 3,
            "processedCount": 3,
            "itemDetail": [
                {
                    "itemIndex": 0,
                    "itemGUID": "5rsIMYlvQECk50YsHsOCiD",
                    "itemState": "SUCCESS",
                    "itemStatusDetail": "",
                    "itemStartDate": "2024-02-01T07:07:10.000+0000",
                    "itemEndDate": "2024-02-01T07:07:12.000+0000",
                    "assetPath": "Explore/BulkPublishAndUnpublish/MyFolder/
Process1.PROCESS.xml"
                },
                {
                    "itemIndex": 1,
                    "itemState": "NOT_FOUND",
                    "itemStatusDetail": "",
                    "itemStartDate": "2024-02-01T07:07:12.000+0000",
                    "itemEndDate": "2024-02-01T07:07:12.000+0000",
                    "assetPath": "Explore/BulkPublishAndUnpublish/MyFolder/
Process11.PROCESS.xml"
                },
                {
                    "itemIndex": 2,
                    "itemGUID": "hib6l1IaggMg9GERey3NcN",
                    "itemState": "ERROR",
                    "itemStatusDetail": "There is/are 1 entry(ies) that should be
unpublished first. Dependent entry(ies): AppConnection1-2",
                    "itemStartDate": "2024-02-01T07:07:12.000+0000",
                    "itemEndDate": "2024-02-01T07:07:13.000+0000",
                    "assetPath": "Explore/BulkPublishAndUnpublish/
ServiceConnectorGoogle.AI_SERVICE_CONNECTOR.xml"

                }
            ]
        }
    },
    "links": {
        "self": http://na1.ai.dm-us.informaticacloud.com/active-bpel/asset/v1/
unpublish/937604908948291875/Status,
        "unpublish": http://na1.ai.dm-us.informaticacloud.com/active-bpel/asset/v1/
unpublish/937604908948291875
    }
}
```

Alternatively, when you send a bulk unpublish request using a request payload, the assets are unpublished in the same order as given in the request payload. You must specify the assets in the order of their dependency to unpublish them simultaneously in the same request.

For example, consider that you have a service connector that is used in an app connection and process, and all the assets are published. You must specify the assets in the following order in the request payload to unpublish them simultaneously:

1. Service connector
2. App connection
3. Process

CHAPTER 5

XML parsing

When you invoke Application Integration assets that contain XML data, Application Integration maintains the XML structure. However, Application Integration might not maintain the same sequence of nodes.

APPENDIX A

HTTP response status codes

The HTTP response status codes are three-digit codes generated by a server in response to client requests. Response status codes serve as a quick and concise means of communicating how the server handled the client's request.

The following table shows the response status codes that Application Integration supports:

Response Status Code	Reason	Description
100	Continue	Indicates that the initial part of a request has been received and has not yet been rejected by the server.
101	Switching Protocols	Indicates a protocol to which the server switches.
102	Processing	Indicates that the server has accepted the complete request but has not yet completed it.
200	OK	Indicates that the request has succeeded.
201	Created	Indicates that the request has been fulfilled and has resulted in one or more new resources being created.
202	Accepted	Indicates that the request has been accepted for processing but the processing has not been completed.
203	Non-Authoritative Information	Indicates that the request was successful but the enclosed payload has been modified from the origin server's 200 (OK) response by a transforming proxy.
204	No Content	Indicates that the server has successfully fulfilled the request and there is no additional content to send in the response payload body.

Response Status Code	Reason	Description
205	Reset Content	Indicates that the server has fulfilled the request and wants the user agent to reset the document view to its original state as received from the origin server.
206	Partial Content	Indicates that the request has succeeded and the body contains the requested ranges of data as described in the Range header of the request.
207	Multi-Status	Conveys information about multiple resources in situations where multiple status codes might be appropriate.
208	Already Reported	Indicates that the members of a Distributed Authoring and Versioning (DAV) binding have already been enumerated in a preceding part of the multi-status response and are not being included again.
226	IM Used	Indicates that the server has fulfilled a client's request for a given resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.
300	Multiple Choices	Indicates that the request has more than one possible response. The user-agent or the user should choose one of them.
301	Moved Permanently	Indicates that the target resource has been assigned a new permanent URI and any future references to this resource must use one of the enclosed URIs.
302	Found	Indicates that the target resource resides temporarily under a different URI.

Response Status Code	Reason	Description
303	See Other	Indicates that the server is redirecting the user agent to a different resource, as indicated by a URI in the Location header field, which is intended to provide an indirect response to the original request.
304	Not Modified	Indicates that a conditional GET or HEAD request has been received and would have resulted in a 200(OK) response if the condition had not evaluated as false.
305	Use Proxy	Indicates a deprecated status code.
306	(Unused)	Indicates a reserved code defined in a previous version of HTTP/1.1, which is no longer used.
307	Temporary Redirect	Indicates that the target resource resides temporarily under a different URI and the user agent must not change the request method if it performs an automatic redirection to that URI.
308	Permanent Redirect	Indicates that the target resource has been assigned a new permanent URI and any future references to this resource must use one of the enclosed URIs.
400	Bad Request	Indicates that the server cannot or will not process the request due to something that is perceived to be a client error such as a malformed request syntax, invalid request message framing, or deceptive request routing.
401	Unauthorized	Indicates that the request has not been applied because it lacks valid authentication credentials for the target resource.
402	Payment Required	Indicates a nonstandard response status code that is reserved for future use.

Response Status Code	Reason	Description
403	Forbidden	Indicates that the server understood the request but refuses to authorize it.
404	Not Found	Indicates that the origin server did not find a current representation for the target resource or is not willing to disclose that one exists.
405	Method Not Allowed	Indicates that the method received in the request-line is known by the origin server but not supported by the target resource.
406	Not Acceptable	Indicates that the target resource does not have a current representation that is acceptable to the user agent, according to the proactive negotiation header fields received in the request content negotiation, and the server is unwilling to supply a default representation.
407	Proxy Authentication Required	Similar to 401 (Unauthorized) but indicates that the client needs to authenticate itself to use a proxy.
408	Request Timeout	Indicates that the server did not receive a complete request message within the time that it was prepared to wait.
409	Conflict	Indicates that the request could not be completed due to a conflict with the current state of the target resource.
410	Gone	Indicates that access to the target resource is no longer available at the origin server and this condition is likely to be permanent.
411	Length Required	Indicates that the server refuses to accept the request without a defined Content-Length.
412	Precondition Failed	Indicates that one or more conditions given in the request header fields evaluated to false when tested on the server.

Response Status Code	Reason	Description
413	Payload Too Large	Indicates that the server is refusing to process a request because the request payload is larger than what the server is willing or able to process.
414	URI Too Long	Indicates that the server is refusing to service the request because the request-target is longer than what the server is willing to interpret.
415	Unsupported Media Type	Indicates that the origin server is refusing to service the request because the payload is in a format that is not supported by this method on the target resource.
416	Range Not Satisfiable	Indicates that none of the ranges in the request's Range header field overlap the current extent of the selected resource or that the set of ranges requested has been rejected due to invalid ranges or an excessive request of small or overlapping ranges.
417	Expectation Failed	Indicates that the expectation given in the request's Expect header field could not be met by at least one of the inbound servers.
422	Unprocessable Entity	Indicates that the server understands the content type of the request entity, and the syntax of the request entity is correct, but it was unable to process the contained instructions.
423	Locked	Indicates that the source or destination resource of a method is locked.
424	Failed Dependency	Indicates that the method could not be performed on the resource because the requested action depended on another action that failed.

Response Status Code	Reason	Description
426	Upgrade Required	Indicates that the server refuses to process the request using the current protocol but might be able to after the client upgrades to a different protocol.
428	Precondition Required	Indicates that the server requires the request to be conditional. This means that a required precondition header, such as If-Match, is missing.
429	Too Many Requests	Indicates that the user has sent too many requests in a given period of time.
431	Request Header Fields Too Large	Indicates that the server refuses to process the request because the request's HTTP headers are too long.
500	Internal Server Error	Indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.
501	Not Implemented	Indicates that the server does not support the functionality required to fulfil the request.
502	Bad Gateway	Indicates that the server, while acting as a gateway or proxy, received an invalid response from an inbound server it accessed while attempting to fulfil the request.
503	Service Unavailable	Indicates that the server is currently unable to handle the request due to a temporary overload or scheduled maintenance, which will likely be resolved after some time.
504	Gateway Timeout	Indicates that the server, while acting as a gateway or proxy, did not receive a timely response from an upstream server it needed to access to complete the request.
505	HTTP Version Not Supported	Indicates that the server does not support, or refuses to support, the major version of HTTP that was used in the request message.

Response Status Code	Reason	Description
506	Variant Also Negotiates	It might be given in the context of Transparent Content Negotiation. This protocol enables a client to retrieve the best variant of a given resource, where the server supports multiple variants.
507	Insufficient Storage	Indicates that the server is unable to store the representation needed to complete the request.
508	Loop Detected	Indicates that the server terminated an operation because it encountered an infinite loop while processing a request with `Depth: infinity`.
510	Not Extended	Indicates that the request did not meet the policy for accessing the resource. The server should send back all the information necessary for the client to issue an extended request. It is outside the scope of this specification to specify how the extensions inform the client.
511	Network Authentication Required	Indicates that the client needs to authenticate to gain network access.

This information is referred from RFC 7231. For more information about the response status codes, see RFC 7231- Response Status Codes.